2

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

A GRAPHICAL USER-INTERFACE DEVELOPMENT TOOL FOR INTELLIGENT COMPUTER-ASSISTED INSTRUCTION SYSTEMS

by

Francius Suwono

September 1993

Thesis Advisor: Neil C. Rowe

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Computer Science Dept. Naval Postgraduate School — 6b. OFFICE SYMBOL *(if applicable)* CS	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS *(City, State, and ZIP Code)* Monterey, CA 93943-5000	7b. ADDRESS *(City, State, and ZIP Code)* Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION — 8b. OFFICE SYMBOL *(if applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State, and ZIP Code)*	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
A Graphical User-Interface Development Tool For Intelligent Computer-Assisted Instruction Systems

12. PERSONAL AUTHOR(S)
Francius Suwono

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Master's Thesis	FROM 08/90 TO: 06/93	September 1993	119

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

An interactive graphical interface helps intelligent computer-assisted instruction systems, because many applications can be well represented by graphic objects. One approach is a facility whereby a teacher constructing a tutor can associate specific graphics with specific predicate-calculus expressions describing a state in a tutoring simulation. This further requires a specification of the arrangement of graphic objects on the screen, how graphic objects can change position with simulation states. It also requires a language for teachers to specify graphic objects. This thesis addresses both. We introduce a broader applications of cartoon animation modelling ideas to tutoring, that have been limited so far by the complexity of their implementation. The special tools provided help computer-inexperienced instructors to develop their own cartoon animation modelling tutor without the need of mathematical description of shapes or activities to be represented. The tutor generator used employes means-ends analysis, and the language for the teachers is built using Prowindows, a Prolog extension for object-oriented design.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Neil C. Rowe	22b. TELEPHONE *(Include Area Code)* (408) 646-2462 — 22c. OFFICE SYMBOL CSRp

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

A Graphical User-Interface Development Tool for Intelligent Computer-Assisted Instruction Systems

by
Francius Suwono
Lieutenant Colonel, Indonesian AirForce
B. S. Aeronautics, Indonesian Air Force Academy, 1969

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

September 1993

Author: [illegible]

Francius Suwono

Approved By: Neil C. Rowe

Neil C. Rowe, Thesis Advisor

[illegible]

Timothy J. Shimeall, Second Reader

[illegible]

Ted Lewis , Chairman,
Department of Computer Science

ABSTRACT

An interactive graphical interface helps intelligent computer-assisted instruction systems, because many applications can be well represented by graphic objects. One approach is a facility whereby a teacher constructing a tutor can associate specific graphics with specific predicate-calculus expressions describing a state in a tutoring simulation. This further requires a specification of the arrangement of graphic objects on the screen, how graphic objects can change position with simulation states. It also requires a language for teachers to specify graphic objects. This thesis addresses both. We introduce a broader applications of cartoon animation modelling ideas to tutoring, that have been limited so far by the complexity of their implementation. The special tools provided help computer-inexperienced instructors to develop their own cartoon animation modelling tutor without the need of mathematical description of shapes or activities to be represented. The tutor generator used employes means-ends analysis, and the language for the teachers is built using Prowindows, a Prolog extension for object-oriented design.

ACKNOWLEDGEMENT

This thesis is accomplished through the help of many people. Very special thanks go to Dr. Neil C. Rowe for allowing me to use his domain independent intelligent tutoring system **METUTOR21** as the basis of this research. His advices during the research process was very valuable. Dr. Timothy Shimeall, my second reader, was critical on giving me direction with the writing process. Mr. Rian from PT. SKILL, Jakarta, helped me with converting my cartridge backup into diskettes. Without his help I would have needed to retype everything. Mr. Toto Tanamas from PT. Istidata Prima, Jakarta, gave me full access to his computers and laser printers inventory, that made it possible to finish up this thesis. Last but not least, thanks to my wife Titiek and my children Tony, Andri and Sylvi for bearing with me through my study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND

Computer-aided instruction (CAI) refers to use of computer to tutor humans [TURB 90]. To a certain extent, such a machine can be viewed as an expert system. However, the major objective of an expert system is to render advice, whereas the purpose of CAI is to teach. Computer-aided instruction has been in use for many years, bringing into the educational process the power of the computer. Now artificial intelligence (AI) methods are being applied to the development of intelligent computer-assisted instruction (ICAI) systems in an attempt to create computerized tutors that shape their teaching techniques to fit the learning patterns of individual students.

An intelligent computer-assisted instruction system basically consists of four major parts: the student modelling module, the expert module, the tutorial module, and an interactive user interface. The expert module is a domain-specific knowledge base where the teacher's expertise and knowledge are stored. The tutorial module, sometimes referred to as the inference engine or tutor generator, is a program which, given a certain response from a student, uses the information in the knowledge base to generate the tutor. The interactive user interface provides communication between the system and the user.

In most ICAI almost all effort is put into building the first three modules, the student modelling module, the expert module and the tutorial module, since they do form the core of the ICAI, and little research effort goes into developing good user interfaces. Yet there are two good reasons not to ignore interfaces issues. First, the best educational software can be ruined by a bad interface, and second, students will not learn to their potential from a program they are reluctant to use [PSOT 88]. Early design of intelligent computer-assisted instruction systems very much depended on natural-language front ends, with human-like dialog. But human-like dialogue, in which the computer prompts the user for answers to a series of questions, is rigid, tedious and machine-centered [HEND 88]. A better approach could be to provide intelligent computer-assisted instruction systems with a graphical user interface, where facts are represented by a set of visual representations (graphical objects) on a graphical display that can be manipulated by the users. The users have no command language to remember beyond the standard set of manipulations and a continuous remainder on the display of the available objects and their states. This direct manipulation approach is good for all intelligent tutors that continuously need to assert and retract part of the knowledge base.

B. OBJECTIVES

This thesis addresses the user interface problem by augmenting a domain-independent computer-assisted instruction system with a graphical user interface and then providing the teachers with a tool for defining their instructional modules to work with the augmented system. We used as basis Professor Neil C. Rowe's domain-independent tutor generator for sequential skills, **METUTOR21** [ROWE 90], which uses means-ends analysis approach. We want to keep the original environment of **METUTOR21** by using the same programming language (**PROLOG**) and /or its extension.The final outcome of this research consists of two parts: the augmented inference engine **MEGRAPH21** and a graphics production tool for the teachers to define their facts in graphics representations, **DRAWGRAPH**.

C. SCOPE

In augmenting **METUTOR21** we followed it closely and preserved its original organization. Some parts, however, need to be completely rewritten to reflect the need of object-oriented design. This graphical user interface improvements are written in **PROWINDOWS**, an extension of the **PROLOG** programming language. A graphics design tool for the teachers comprises the other half of the effort in this thesis. It provides a way to define graphical objects, to specify their position on the interface screen, and to define the interaction between graphical objects and the tutored states.

D. ORGANIZATION

Chapter II of this thesis describes previous works on ICAI, some of them with graphical user-interface. Chapter III describes in a considerable detail the **METUTOR21** as a domain-independent tutor generator. Chapter IV provides the description of our programming to provide the tutor generator with graphical user interface. Chapter V discusses the results and summarizes the performance of this thesis research using fire-fighting tutor **MEFIRE** [ROWE 90] as the teacher module. Chapter VI closes the thesis with an overview of the work accomplished and areas for future research.

II. PREVIOUS WORKS ON INTELLIGENT COMPUTER-ASSISTED INSTRUCTION SYSTEMS

Many intelligent computer-assisted instructions had gained popularity and shown impressive teaching performance. Programs such as IMTS [PSOT 88], OASIS [HEND 88], and SOHPIE [SLEE 82] are considered as successful ICAI programs, but all of them are highly specialized or domain-specific, so specific that has made it difficult to extend them to related domains. **METUTOR21** that employs means-ends analysis is capable of surprisingly powerful reasoning, yet is easy to customize for an application. It has been successfully applied to a wide range of training tasks, some of them have been for fire-fighting tutor on naval ships [WEIN 88], cardiopulmonary resuscitation [CAMP 88], network-mail program [KIM 88], replenishment of ships at sea [SALG 89], and pilot emergency tutoring system [KANG 90]. However, no one has tried tieing graphical user interface to **METUTOR21**. The followings are description of examples of related tutoring programs, some of them have graphical user interfaces.

A. INTELLIGENT MAINTENANCE TRAINING SYSTEMS (IMTS)

The Intelligent Maintenance Training System (IMTS), was developed by Behavioral Technology Laboratories, University of Southern California, in early 1985 [PSOT 88]. The major objectives of the IMTS are to construct a cohesive maintenance training system in a functioning systems, and to produce an operational maintenance training system that can be used by instructors to meet a wide range of training needs. The first application of the IMTS will be in training corrective maintenance of the SH-3H helicopter's bladefolding subsystem. The system was envisioned from the start to operate on off-the-shelf processing and display hardware manufactured in quantity and maintained commercially, and mouse is used for computer-student interactions that rely heavily on simulation of the real system. The system includes an *object construction* editor to define the graphic appearance and functions of generic objects, a *system construction* editor to build a specific system from the generic objects, and a *simulator* to determine the symptom of the faulty part. After a new object is defined graphically it is given rules for its behavior using a *generic-object behavior* editor.

The IMTS screen is divided into four major sections for display. The fixed view of the system organization provides the student with a means of selecting close-up views of system subsections. The text area presents verbal messages from the IMTS in response to student actions, or as part of a guided simulation. The main simulation display area shows a detailed diagram of one portion of the system. The convenience viewing area displays copies of some of the system elements that appear in the detailed diagram. Each practice problem consists of a malfunction, inserting the malfunction into the simulation data of the system, initializing the control settings, and displaying the operator's complaint in the text area. The student uses the mouse to directly manipulate simulated objects and to select menu commands and other options. Actions are performed by selecting the portions of the system diagrams containing controls, indicators and test points of interest, and manipulating the displayed elements as if they were the real equipment.

IMTS can be configured for new training applications, but it consists almost entirely describing the new architecture to the system.

B. QUEST

QUEST (Qualitative Understanding of Electrical Systems Troubleshooting) is an ICAI system for teaching electrical system troubleshooting. The system employs qualitative simulation models to produce explanations and to generate animated displays about the behavior of simple electrical circuits. The simulation includes a description of the circuit with rules for evaluating states of devices. It supports a dynamic presentation environment using graphics. Troubleshooting concepts and strategy within the environment of circuit operation can be demonstrated by the expert troubleshooting program. QUEST diagnostic feature elicits explicit information from the student about the intended purpose of his or her actions before they are performed and about his or her conclusions afterward. The interface is relatively easy: the student answers questions by choosing from a range of responses from the display window using the mouse.

C. GUIDON

GUIDON [CLANC 87] is a structured, case-method-dialog teaching program. The program generates teaching material from rules, constructs and verifies a model of what the student knows, and explains expert reasoning. The objective of GUIDON is to convert MYCIN, an infectious-disease knowledge-based consultation program into an instructional .tool. As an interactive teaching program, GUIDON systematically compares student behavior to what MYCIN would do, alerting the student to discrepancy in a consultation. The explanation program in GUIDON abstracts key reasoning steps and presenting prosaic summaries and subgoal diagrams. GUIDON has been coupled to other knowledge-bases like the PUFF consultation system for diagnosis of pulmonary function and the SACON system for structural analysis. The purpose of a GUIDON tutorial is to use the context of an actual problem to make the student aware of inconsistencies in his knowledge, and to correct these inconsistencies. The student-computer communication is limited to a computer terminal that print one line at a time, graphical representations are not considered.

D. SOPHIE

The Sophisticated Instructional Environment (SOPHIE) [SLEE 82] is a pioneering ICAI system for electronics troubleshooting training. It used a general circuit simulation program as a dynamic knowledge base for evaluating the behavior of the circuit under working or faulted conditions. The student goal is to find a fault in the simulated system. SOPHIE uses device-base simulation to support the checking of student inference, as well as heuristic strategies for question generation and answering mechanism. The understanding capabilities in the system was based on its use of a general purpose circuit simulation called SPICE, together with a LISP-based functional simulator. SHOPIE used the simulator to make powerful inferences about circuit behavior. It could infer what the student should have been able to conclude, however it could not tell whether the student's conclusions were based on logically complete and consistent reasoning.

E. STEAMER

STEAMER [KEAR 87] is an intelligent computer-assisted instruction system that simulates a ship's steam propulsion plant. It is used to train operators by helping them understand the complex system through interactive graphical interfaces. The actual implementation of reasoning mechanism is purely mathematical, and it has inspired research in the areas of mental model and abstraction simulation in AI.

III. METUTOR21 TUTOR GENERATOR

METUTOR21 is a domain-independent inference engine used for teaching sequential skill [ROWE 90]. This reasoning mechanism implements means-ends analysis method that is similar to the way people solve many sequence planning problems. Unlike other tutor programs that teach sequential actions, **METUTOR21** is easy to customize for an application. It requires only assertion of preconditions, postconditions and a set of recommended conditions for actions. The simple method can easily be understood by people not expert in artificial intelligent, making it easy to construct a tutor program.

A. MEANS-ENDS TUTOR

Means-ends analysis is a classic technique for solving search problems by abstraction. The top-level of means-ends analysis in **METUTOR21** is a recursive means-ends predicate with four arguments: **State, Goal, Oplist,** and **Goalstate**. The **State** is a complete list of true statements (facts) in a state, the **Goal** is a list of facts that are required in the goal state, the **Oplist** is a list of operators required to reach the goal state from the starting sate, the **Goalstate** is a complete list of facts at the goal state.

The means-ends tutoring strategy checks if the student action agrees with what the tutor think is the best action. If it doesn't the student is tutored and shown the implications of his or her action. Six domain-independent bug types can be detected with student-selected actions [ROWE 88]:

1. The student's action is a misspelling of a known action.
2. The student's action is easily confusable with a recommended action.
3. The student's action does not satisfy preconditions of the action.
4. The student's action is an irreversible one: that means the problem can never be solved if the action is performed.
5. The student's action is useless.
6. Both the student's action and the computer-selected action are possible in some state

To know what operators to apply for a specific situation, means-ends analysis needs to know the difference between the current state and the goal state. This difference usually is stored in a difference table. Using the recommended operators the difference table recommends an operator appropriate to a state and a goal. It is important to note that the difference table only recommends, with no concern if the operator can actually be applied to the state. The recommendation can be overruled by the stronger precondition requirements. Through the difference table a problem is decomposed into three simpler subproblems: a subproblem of getting from the current state to an intermediate state in which the student can apply the recommended operator, a subproblem of applying the operator and get to a new state, and a subproblem of going from there to the goal state.

The last two steps are search problems themselves, possibly requiring additional decompositions of their own. Means-ends analysis is also hierarchical reasoning because it starts with big problems and tries to gradually reason down to little details [ROWE 88]. As shown in the decomposition process above a complete and correct specification in the difference table of preconditions and postconditions of operators is essential for means-ends analysis.

B. KNOWLEDGE REPRESENTATION IN METUTOR21

METUTOR21 is written in **PROLOG** [QUIN 90] using predicate calculus expressions as the format for its knowledge representation. The teacher defines the recommended conditions, preconditions, deletepostconditions and addpostconditions rules and optional facts in the knowledge base as the description of the domain.

1. Recommended Conditions

Recommended conditions are collection of facts about what action is recommended to get to a certain condition. Predicate **recommended** with two arguments:

recommended(Operator, Difference-list)

shows what operator to apply to get a listed difference between goal state and current state. Recommendations are weaker than preconditions because they don't take into account other conflicting conditions. A recommended condition may not be applicable if the precondition does not support it.

2. Preconditions

The **precondition** predicate gives the prerequisites for applying an operator. This is a stronger condition for applying an operator than the recommendation. The recommendation only recommends, with no consideration whether the operator can be applied or not, whereas **precondition** requires that the prerequisites are met before an operator can be applied.

3. Deletepostconditions and Addpostconditions

The **deletepostcondition** predicate lists conditions deleted when an operator is applied. These conditions are no longer true after the operator is applied and need to be removed from the state. The **addpostcondition** predicate lists conditions added into the difference list (list containing the difference between the current state and the goal state) when an operator is applied. These conditions become true after the application of the operator, and are added to the state.

4. Optional facts

The following facts are optional and can be added in the teacher's definition of the knowledge base. The **randsubst** predicate gives random-substitution triples or quadruples, each with the arguments: initial-fact, ending-fact, transition-probability, and message to user. It gives additional changes beyond those specified in the postconditions to simulate a real-world situation and to challenge the student. The **noprefs** predicate shows that the order of priority of two operators in the 'recommended' rules is arbitrary. The **intro** predicate gives introductory information for the student. **Debugflag**, if asserted, prints additional debugging data. **studentflag**, if asserted, prevents the program from checking for teacher errors, so the execution is speeded up.

C. THE USER INTERFACE OF METUTOR21

METUTOR21 uses natural language as the medium of communication between the system and the user. The built-in natural language processor inside **METUTOR21** "works most of the time", as Professor Rowe termed it, however it is limited. The natural language processor translates typed-in input from the user into a predicate expression that can be used by the tutoring system for the reasoning process, and translates the resulting predicate expressions into natural language to be displayed to the user.

No special window is designed for interactive communication between the student and the tutoring system. **METUTOR21** runs in any **PROLOG** window. Help is provided showing the list of possible operators in randomized order. Some filtering on the student typed-in input such as spelling correction and checking of student confusion is provided. As mentioned in the introduction, natural language front ends like this in which the computer prompts the user for answers, is rigid and machine centered. At best, it is error-prone and it distracts the students from the learning contents, thus reduces its effectiveness. Improving **METUTOR21** with direct manipulation user interface would overcome this problem.

IV. IMPROVEMENT ON METUTOR21 FOR DIRECT-MANIPULATION USER INTERFACE

Direct-manipulation user interfaces need to represent menus and selections as objects. This is possible for programs written in **PROLOG** with **PROWINDOWS** [QUIN 88], an object-oriented extension of **PROLOG**. Quintus **PROWINDOWS** is an object-oriented programming package that enables **PROLOG** programmers to create window-based user interfaces for their application programs.

A. THE OBJECT-ORIENTED APPROACH IN PROWINDOWS

PROWINDOWS uses objects and messages between objects to create window-based user interfaces. These high-level messages are passed between **PROWINDOWS** and **PROLOG**, with **PROWINDOWS** handling the low-level tasks required by the program [QUIN 88]. The tasks that **PROWINDOWS** objects can perform are called the object's behaviors.

The objects in **PROWINDOWS** are grouped into classes, with bottom-level objects called instances of a class. *Kernels* are classes that describe objects, messages, and other classes. *Data Types* are classes that describe abstract data types, including points, dimensions, collections of objects, and lists. *Windows* are classes that provides access to most of the facilities of the window system such as creation, display and destruction of widow families. *Dialogs* are classes that allow the user to directly communicate with an application program by using various kind of menus, buttons, and the keyboard. *Texts* are classes that provide simple text manipulation tools for loading, editing, and saving text. *Graphics* are classes that describe both primitive graphic objects (bitmaps, boxes, circles, ellipses, lines, paths, text and text-block) and compounds of primitive graphic objects. All primitive graphic objects can be displayed inside a window class, especially subclass *picture*. The basic entity which can be displayed in a picture is called a *graphical*. Messages to these *graphicals* can be sent through programs or direct interactive using a mouse or typed-in input through a dialog window.

B. ASSOCIATING NATURAL LANGUAGE WITH OBJECTS

Typed-in text inputs are no longer used in **MEGRAPH21**, our augmentation of **METUTOR21**. Instead, users use the mouse to select the operators they want to apply from a list displayed in a browser window. This approach requires that operators be converted into *text objects* that can be displayed in random order. Our first step to do this is to create a browser window, where the operator objects are displayed. A browser window is a special window object for displaying a list of selection objects. Objects inside a browser window are arranged vertically, and can be selected by clicking the mouse left button.

Our next step is to find all possible operators. On initialization we extracted the operators from the second arguments in the **recommended** rules, and convert every one of them into a *string object*. The browser window shows the list of operators, and the operators are now objects with their own behaviors. This window is a scrolling window, which means that the amount of visible space does not limit the number of operators displayed. This kind of menu is favored for displaying operators that vary in number from one tutorial program to another.

The natural language to text object conversion is handled by the predicate:

showlist(Text, L, R)

with Text being the text object, L the predicate expression list, and R the type of the expression (in this case operator type). We also prepare a translation table in the form:

translate(predicate_form, natural_language_form)

to avoid operator processing every time a translation is needed.

The conversion from fact predicates and preconditions into text objects is handled in a similar way. The predicate expressions are converted into natural-language-like string objects and then sent to a *text-block* object.The text-block object is in turn sent to the display window. This is because the text-block object can arrange the text in an intelligent way. It can align the text according to the teacher's specification, by left, right or center alignment. The natural-language format for the output objects follows the original format in **METUTOR21**.

There is a conversion from fact predicates to filenames for most fact predicates, which is basically the same operation as the conversion operation above. The difference is that instead of blank space between words we put underline in the filename conversion. This conversion is used for saving and calling a bitmap file.

C. SCREEN LAYOUT OF MEGRAPH21

The **MEGRAPH21** interface screen is of size 930 by 850 pixels (see Figure 1), taking almost the whole screen in a Solbourne S4000 workstation. This is done to eliminate distractions due to a crowded screen display. There are two main parts of the window, the large display window itself on the left, and the operator selection browser on the right.

The topmost of the main display window gives the problem context. It introduces the tutor to the student as to give a general idea of what is happening. Teachers are encouraged to provide this intro in their tutorial definition, and otherwise the 200 characters space provided is not used. Next is the objective definition. This objective is the final goal the student should achieve. This can be considered as the definition of *success*, and explicitly stated by the teacher in the predicate **goal**. Four lines (approximately 400 characters) are provided for this. If the space is exceeded some of the characters will not be displayed. Below this is the current state definition, the condition at this point in the simulation. At the onset of the tutoring, this state uses the teacher definition of **start_state** and it is changed dynamically during the session. The same amount of space as the objective is provided for this display.

A big box under the current state display is the graphical presentation of the state described just above. This seems like a redundancy, but it has a point. The two different formats, textual and graphical, each have advantages and disadvantages. Textual representation can convey a meaning difficult to describe graphically like the state of an object, but graphical representations make the problem states look more real.

Under the graphical representation box is the student selection box. It shows the last operator selected by the student in a small box. Below this box is the tutor display, displaying whatever the tutor says or comments. This comments are made by the inference engine as the tutoring progresses.

Except the top two parts, everything in the main display window is changed with changes to the tutoring state.

D. GRAPHICAL REPRESENTATION OF STATES

Representing states graphically requires that every fact object has a corresponding graphic representation. Better yet, the mapping should be from sets of facts to a graphic object, considering that one fact can have several states, for example a team can be at the repair locker or at the fire and can either be equipped or unequipped.

The teacher is responsible for defining the contextual combinations of facts for graphical representation. For example the fact **fire is raging** must be represented differently according to the location of the fire team. The tutor generator will check the state for a contextual combination, and then display the associated representation if any, otherwise the default representation will be displayed.

The default representation is stored with the same filename as the fact. The contextual combinations are defined by the predicate **display** with three arguments. The first argument is the fact to be represented, the second argument is the list of additional facts providing context, and the third argument is the filename of a graphical representation of the fact. The following examples show how contextual combinations are defined:

```
display(raging(fire), [location(repair, locker)], fire1)
display(raging(fire), [location(fire)], fire2)
display(smokey, [location(repair, locker)], smokey1)
display(smokey, [location(repair, locker)], smokey1)
```

display(smokey, [location(fire)], smokey2)

display(location(repair, locker), [], team1)

display(location(fire), [], team2)

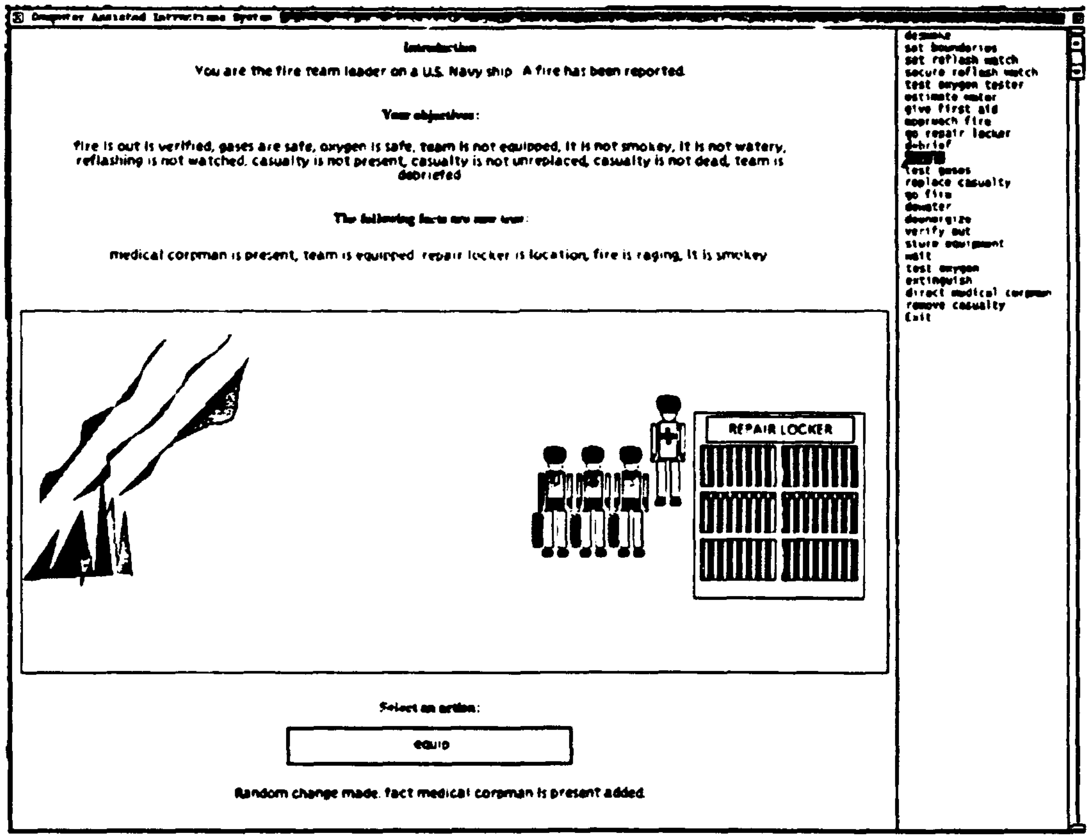
display(approach(fire), [], team3)

display(equipped(team), [location(repair, locker)], equipment1)

display(equipped(team), [location(fire)], equipment2)

display(equipped(team), [approach(fire)], equipment3)

These definitions must be included in the teacher's module. Figure 1 and Figure 2 show the implementation of this approach in our session window with the above contextual definitions inserted in **MEFIRE**.



Introduction

You are the fire team leader on a U.S. Navy ship. A fire has been reported.

Your objectives:

fire is out is verified, gases are safe, oxygen is safe, team is not equipped, it is not smokey, it is not watery, reflashing is not watched, casualty is not present, casualty is not unreplaced, casualty is not dead, team is debriefed

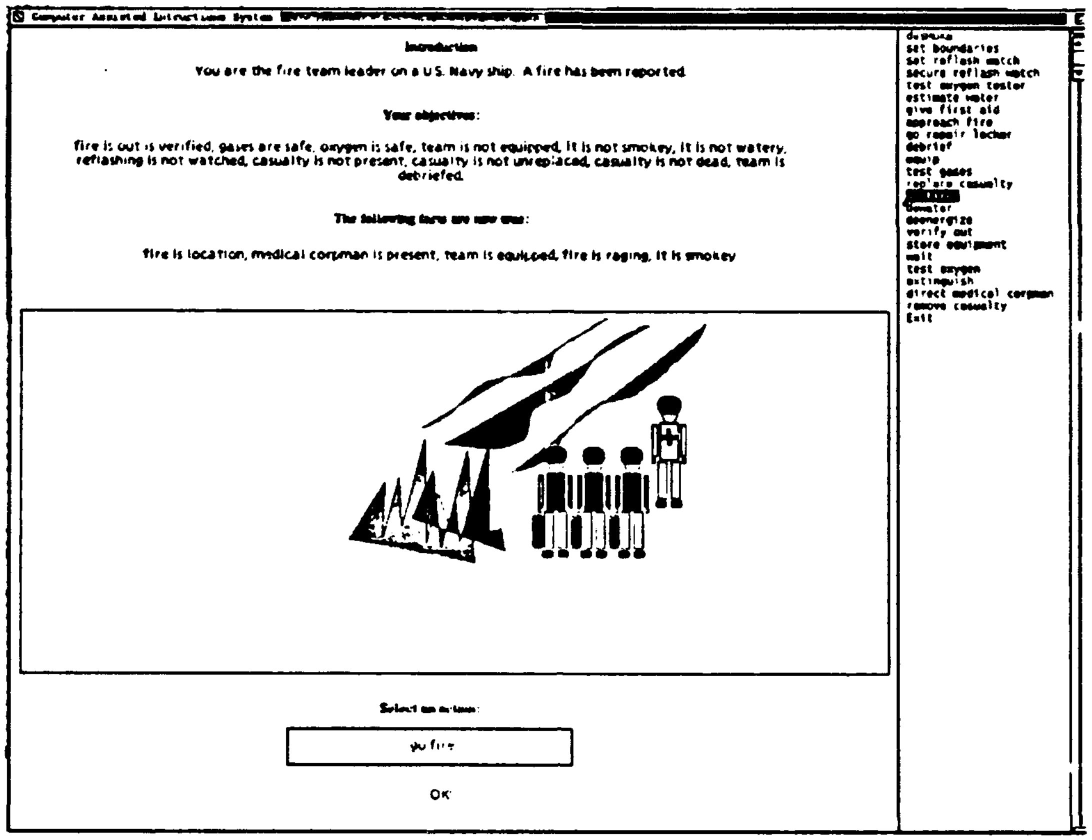
The following facts are now true:

medical corpsman is present, team is equipped, repair locker is location, fire is raging, it is smokey

Select an action:

Random change made. fact medical corpsman is present added.

Figure 1: MEGRAPH21 session window showing location is repair locker.



Introduction

You are the fire team leader on a U.S. Navy ship. A fire has been reported.

Your objectives:

fire is out is verified, gases are safe, oxygen is safe, team is not equipped, it is not smokey, it is not watery, reflashing is not watched, casualty is not present, casualty is not unreplaced, casualty is not dead, team is debriefed.

The following facts are now true:

fire is location, medical corpman is present, team is equipped, fire is raging, it is smokey

Select an action:

Figure 2: Session window showing change of state from Figure 1. Location is fire.

The graphics display area is refreshed every time the state changes to display the new state. To prevent refreshing the whole window, the graphic display area is isolated from the rest of the window area. This is done by using a bitmap object as the displaying area. This bitmap object is of size equal to the graphics display box. It is appended to the main window. Whenever the state changes, **MEGRAPH21** changes the bitmap instead of refreshing the whole display window. Each drawing displayed on the graphic display box, and associated with a set of facts, is loaded from bitmap files created by the teacher using **DRAWGRAPH**, a graphics tool.

E. DEFINING GRAPHICAL FACT OBJECTS

Since **MEGRAPH21** can only call graphic objects that previously stored in **PROWINDOWS** format, we cannot just have the teacher uses any graphics tool like FrameMaker to create the pictures needed. Also, many graphics tools are too complex for computer-naive teachers to use. So we created **DRAWGRAPH** the graphics tool for defining graphical objects that goes with **MEGRAPH21**. Using DRAWGRAPH the teacher can create new drawings, group several drawings into one object, ungroup an object into individual graphic objects, make copies of a graphic object, save and display previously saved drawings.

1. Drawing Creation

Our graphics editor **DRAWGRAPH** uses primitive graphic objects such as lines, paths, boxes, circles and ellipses as building blocks to create an object. The graphic primitives permit the creation of many graphic images. An additional class called *figure* is provided in **PROWINDOWS** to enable the user to interact with graphic objects. Figures are objects having three special capabilities:

1. They can represent a collection of graphicals, within a fixed reference frame.
2. They can be moved either with a mouse or under program control.
3. They can be linked to one another to represent a logical relationship.

All graphic objects created with **DRAWGRAPH** are appended to a figure the first time they are created, so they can be manipulated by the teacher (see Figure 3). However, we chose to save it in a bitmap, because it is the simplest way of saving. The problems with saving in a bitmap are the required large amount of space and that we can not manipulate the individual objects anymore, since a bitmap is a collection of pixels.

2. Grouping and Ungrouping Features

To represent complicated objects, **DRAWGRAPH** is equipped with a grouping capability. Several graphical primitives can be grouped into one object to make it easy for further manipulation like copy, move etc.

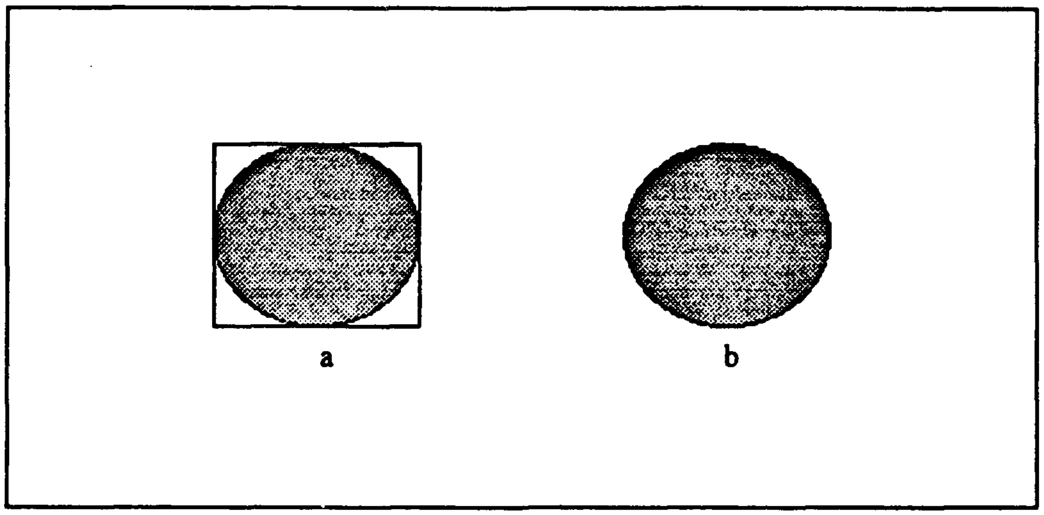


Figure 3: A circle primitive object on its figure (a) and direct display (b).

The objects selected for grouping are separated from their initial figures, and then the graphical primitives are all appended to a new figure (see Figure 4). A position correction is needed in this process because the graphical primitives still hold an initial position which does not conform to the group figure position. Ungrouping is a less complicated procedure. The selected object is first separated from the figure to which they were appended, and each individual graphical object is appended to its own new figure.

3. Copy Feature

DRAWGRAPH's copy feature makes a duplicate of an object, whether they are primitives or grouped objects. A duplicate object is initially displayed at five pixels down and five pixels right of the original object.

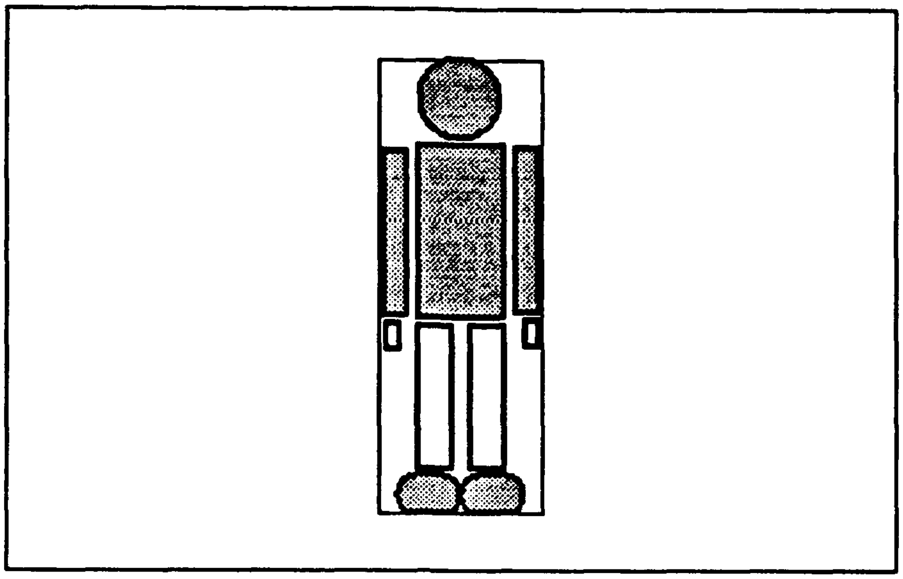


Figure 4: Grouped primitives make one object.

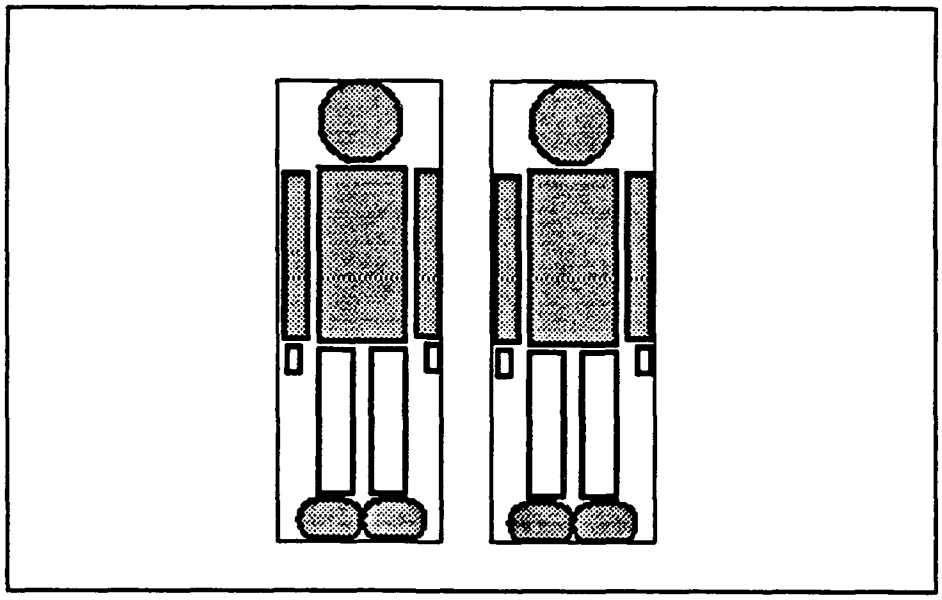


Figure 5: A copied object.

4. Saving Objects

Saving an object is the last step in defining a graphical object. The save option from the main menu will display a submenu with four buttons: Clip, Cut&Save, Done and Cancel. To save an object the teacher must define first the area to be saved using the Clip option. The Clip option allows the teacher to draw a box surrounding the area to be saved.

It is important to draw the box as close as possible to the object to cover the smallest area possible, so the bitmap file where it will be saved is minimum. After an area is defined, we use the Cut&Save button to save the clipped area. Cut&Save option will give us the list of possible filenames that we can use. Selecting one of the filenames and clicking the OK button will save the clipped area to a file of that name. Filenames other than the ones offered by the listing can also be defined by typing in the text area. Since a bitmap does not retain any information about position, the graphic object position is saved in a text file called **posfile**. This file is created and maintained (add or delete) by **DRAWGRAPH** and contains facts in the following format:

area(filename, X, Y, Width, Height).

This file is loaded by **MEGRAPH21** to position the graphic objects in the display area, and used by **DRAWGRAPH** in the display option.

5. Display Option

The display option will display previously defined graphic objects in the drawing area. It can be used to position the next drawing relative to the previous drawing. While the displayed objects occupy the same space with the newly created object, they are appended to different window-class object. The displayed object is appended to a *bitmap object*, while the new object is appended to a *picture object*. The save option ignores the bitmap object in the clip area (see Figure 6). Using this approach the exact position of graphic objects can be defined. The drawing area is of the same size as the **MEGRAPH21** graphically area, hence position in drawing is important. It will be the same as on the tutor.

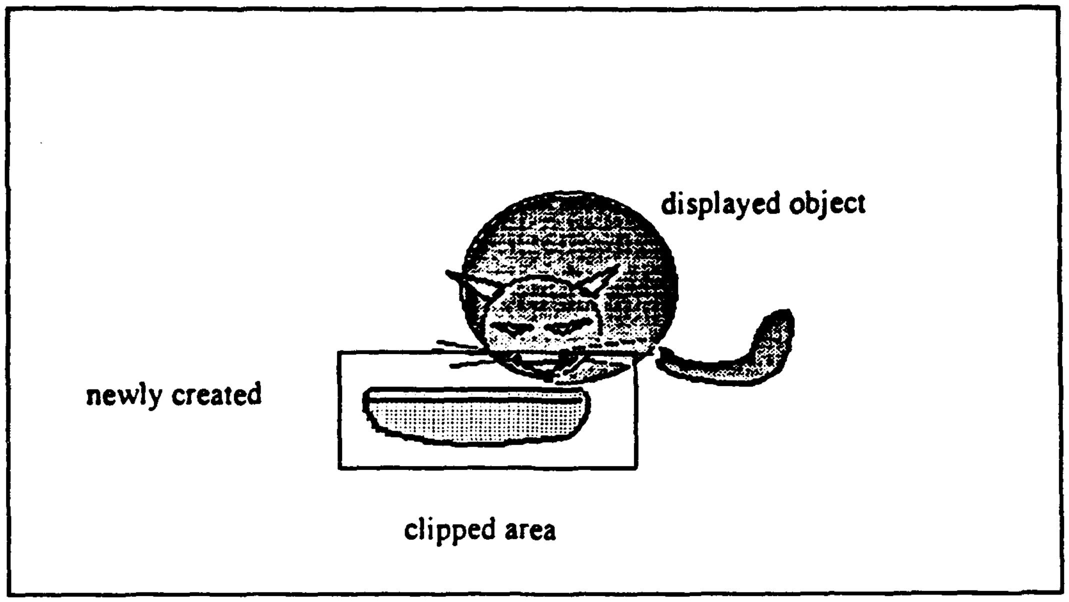


Figure 6: The newly created object (bowl) will be saved, the cat will not.

V. DISCUSSION OF RESULTS

We tested the tutoring system first without any graphical display, and then with graphical representation completely defined. For this testing we used MEFIRE [ROWE 90], the fire fighting tutor, as the tutorial module.

A. PROGRAM SIZES

Files	Description	Source Code Size in bytes
MEGRAPH21	Tutorial module	37,322
DRAWGRAPH	Graphic editor module	30,732
COMMON	Common module, used by both MEGRAPH and DRAWGRAPH	15,068
MEFIRE	The fire-fighting tutor module	8,515
POSFILE	File for storing graphics' position	1,071
	Total	92,708

TABLE 1: **Program sizes.**

The sizes of the programs are shown in Table 1. The total size is approximately 93 kilobytes including the tutorial module which varies, depending on the teacher's application. Besides the program files there are bitmap files that store the graphical representation of facts defined by the teacher. As shown in Table 2, some of the sizes are big due to the bitmap files. That is why it is important to clip the file as close as possible to the graphical objects to reduce the storage size. The sizes will vary from one application to the other, and depend on how complicated the representations are.

Bitmap Filename	Operator applied	Size in bytes
boundaries_are_set	set boundaries	19,080
casualty_is_present	random substitution,	9,345
casualty_is_treated	give first aid	5,364
fire_area_is_deenergize	denergize	5,324
fire_is_confronted	go fire	25,957
fire_is_out_is_verified	verify out	8,007
fire_is_raging	starting state	16,756
gases_are_safe	none	11,106
gases_are_tested	test gases	13,518
it_is_smokey	starting state	35,464
it_is_watery	starting state	11,157
medical_corpman_is_present	random substitution	6,441
oxygen_is_safe	none	14,475
oxygen_is_tested	test oxygen	13,656
oxygen_tester_is_tested	test oxygen tester	10,897
reflashing_is_watch	set reflash watch	15,628
repair_locker_is_location	go repair locker	58,639
team_is_debriefed	debrief	18,229
team_is_equipped	equip	14,287
team_is_not_equipped	store equipment	13,399
water_is_estimated	estimate water	3,126
	Total	332,855

TABLE 2: Graphic representation bitmap sizes from MEFIRE.

Trial	Mem Total In Use (bytes)	Program Space (bytes)	Global Stack (bytes)	Local Stack (bytes)	Runtime in seconds
1	1306144	454228	719836	624	62.583
2	1276672	490288	612060	624	94.316
3	1206704	420320	645660	624	51.816
4	1165908	445056	583432	664	73.733
5	1167558	446596	551792	624	96.033
6	1249036	462652	639940	624	125.116
7	1255152	468768	673720	624	149.833
8	1198448	477596	591476	624	177.566
9	1202504	481652	610204	624	204.916
10	1208356	487504	580036	664	228.583
11	516048	384984	31492	416	41.033
12	1191876	405492	623248	624	62.833
Avg.	1064875	452095	571908	613	114.030

TABLE 3: **Runtime and memory usage statistic for MEGRAPH21 using MEFIRE.**

B. RUNTIME

The runtime of MEGRAPH21 using MEFIRE as the tutorial module is described in the following statistics. The runtime and the memory usage vary according to the tutoring. More mistakes made by the student means more runtime and more memory usage, and no mistakes takes the minimum runtime and memory usage. Time also depends on whether random substitutions are in effect. We ran the program 12 times, the minimum runtime (CPU time) was approximately 41 seconds and the maximum was 229 seconds, with the average runtime of 114 seconds (see Table 3). The maximum, minimum and average of memory in use, program space, global stack and local stack are shown in Table 4.

Memory	Maximum (bytes)	Minimum (bytes)	Average (bytes)
Total memory in use	1.306.144	516.048	1.064.875
Program space	490.288	384.984	452.095
Global stack	719.836	31.492	571.908
Local stack	664	416	613

TABLE 4: **Maximum, minimum and average of memory usage.**

C. ACCURACY

We were concerned with the accuracy of the graphical representation displayed on the screen and the list of facts displayed in the state text. Throughout the testing we found that the list of facts was represented as intended accurately. The position of graphical objects on the display screen was the same as when they were created. Even when the student made mistakes, the change of state was exactly represented by the graphical representation. We simulated several mistakes during the testing, and no deadlock was detected.

VI. CONCLUSION

A. MAJOR ACHIEVEMENTS

This thesis is the first attempt to provide a graphical user-interface to **METUTOR21**. Two major achievements were accomplished: First, we provided the domain-independent tutoring system with a direct manipulation interactive user interface; second, we provided the teachers with a tool to define their facts in a graphical representation.

In those respects, **MEGRAPH21** is innovative in the following ways:

a. It introduces a broader applications of cartoon animation modelling ideas to tutoring, that have been limited so far by the complexity of their implementation.

b. The special tool provided helps computer-inexperienced instructors to develop their own cartoon animation modelling tutor without the need of mathematical description of shapes or activities to be represented. Image can be created and location is specified, and association with particular facts or set of facts can be defined. The correspondence of facts to images can be specified not only as a single fact to a single image, but also multiple facts to a single image or multiple facts to multiple images.

B. WEAKNESSES AND RECOMMENDATION

The facts, objectives and teacher-comments display of 400 characters in **MEGRAPH21** may not enough for bigger programs, and scrolling will be necessary. 'Greying' the previously selected operators will help the students see the rest of the operators that is not applied yet. For better animation more than just contextual relation definition of facts is needed. We need more graphical definitions and integration management in the tutor module.

Screen refreshing should not be necessary every time there is a change in state. Instead, an individual graphical representation should be asserted or retracted from the display screen. Every graphical object should be editable. This requires that the drawings be stored as graphical objects as well as bitmaps. The ability to read bitmap files stored in format other than **PROWINDOWS** format would be an advantage.

Other drawing features can be added. Free-hand drawing capability can enhance the drawing flexibility. Rotate should provide 90 degree rotation of graphical objects. Scale should scale an object size. Front/Back should move an object display in front or in back of other objects. Flip Up/Down should rotate an object 180 degrees. Polygons and arcs can be added to the graphical primitives. A library of shapes would help teachers with non-artistic inclinations.

APPENDIX A

DRAWGRAPH USER'S GUIDE

Introduction and Purpose.

DRAWGRAPH is a graphic tool specially designed for defining graphical objects to be used in intelligent computer-assisted instruction systems. The graphical objects are created using primitive objects such as lines, paths, boxes, circles, ellipses etc. as well as free-hand drawings. Through a menu selections the line thickness of the drawing can be defined. Users can also define whether or not the line will have an arrow end. The primitive objects are movable for arrangement by click-and-dragging the mouse middle-button.

DRAWGRAPH derives filenames into which the objects will be saved from the teacher's tutoring program, the program where all rules concerning the tutored skill are defined. This requires that the tutoring program be created first before attempting to run **DRAWGRAPH**. These filenames represent facts in the tutoring program and are displayed in a browser menu, which can be selected by the users to name the object's files. The users also have the freedom to make a filename other than the one offered by **DRAWGRAPH** by typing it. The users specify part of the drawing that will be saved using 'clip' and 'cut-and-save' features, and the drawing objects are saved as bitmap files.

DRAWGRAPH is written in Prowindows, an object oriented graphic language extension of Prolog, to complement the teachers tutoring program which is written in Prolog. **DRAWGRAPH** compiles in Prowindows under Unix X-Window environment. The users of this tool are teachers building tutoring programs intended to run on the augmented Professor Rowe's intelligent computer-assisted instruction system using means-ends analysis, **MEGRAPH21**. The users are assumed to be familiar with Prolog and able to write tutoring programs for sequential skills.

Products of DRAWGRAPH sessions

There are two kind of files created during a **DRAWGRAPH** session:

a. Object files, bitmaps files that contain the objects created during a **DRAWGRAPH** session. These are objects defined by the teacher for the tutoring program. For the sake of simplicity all object files created during **DRAWGRAPH** session will reside in the same directory as the tutoring program.

b. **posfile**, a file containing the information about the area the objects will occupy when they are displayed. The area information contains the filename, xy-coordinates of upper-left corner of the area, width and height. This information needs to be stored separately because bitmaps do not retain them.

DRAWGRAPH Menu Organization

DRAWGRAPH's drawing plate is of size 910 by 384 pixels (see Figure 8) and will open together with a dialog window that contain the selection menu. The menu lay out detail is shown in Figure 8.

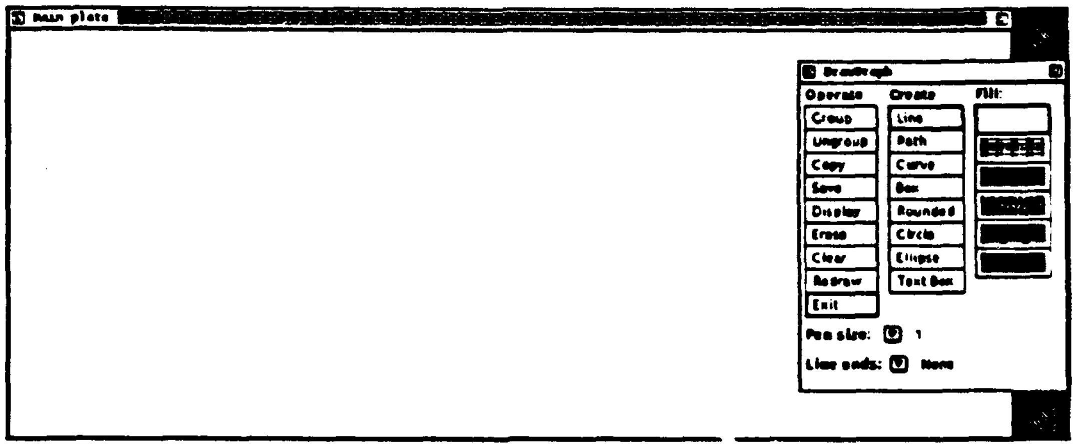


Figure 7: DRAWGRAPH main-plate and the menu

1. Menu for Operate

- Group: to group several selected objects into one object.
- Ungroup: to separate a group of objects into individual objects.
- Copy: to duplicate an object.
- Save: to save a drawing object as a bitmap file.
- Display: to load a bitmap object and display it in the drawing plate.
- Erase: to erase an object.
- Clear: to erase all drawing objects from the drawing plate.
- Redraw: to refresh the screen if the drawing is messed-up.
- Exit: to exit the **DRAWGRAPH** session.

2. Menu for Create

Menu for Create cons^i^ts of the following buttons:

- Line: to draw straight ıines.

- Path: to draw straight paths.

- Curve: to draw smoothed paths.

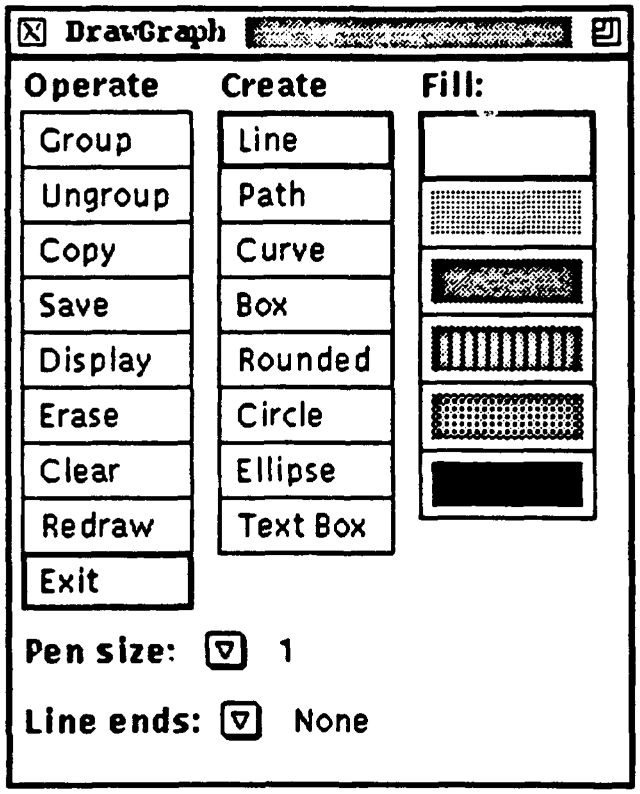
- Box: to draw boxes.

- Rounded: to draw boxes with rounded corners.

- Circle: to draw circles.

- Ellipse: to draw ellipses.

- Text Box: to draw string of characters inside a box.



Pen size: 1

Line ends: None

Figure 8: DRAWGRAPH menu

3. Menu for Fill

The 'Fill' menu consists of several shades of fill patterns, including black and white with white fill as default. Once a fill is selected by clicking the left mouse button, it can be used to fill several objects without the need of going back to click the fill, until a different menu is selected.

4. Pen Size

'Pen size' is a cycle menu, consisting of ten pen size selections. The pen size indicates the thickness of a line in pixels. Pen of size one is the default. Using a pen size bigger than half the area of an object will fill the object with black.

5. Line Ends

'Line ends' is also a cycle menu, consists of four selections: 'None', 'First Arrow', 'Second Arrow', or 'Both Arrows', which will put arrows at line ends according to the menu selection. Note that 'First' means the start of a line, and 'Second' is the end of a line when it is created.

Starting DRAWGRAPH

DRAWGRAPH requires that Prolog and Prowindows are properly installed, and can be called from **DRAWGRAPH** directory. At Prowindows prompt load the following files:

- **DRAWGRAPH**
- **common**, a utility file, and
- the teacher program.

This can be done by typing '[drawgraph, common, 'program name'].'. After a successful loading, again at Prowindows prompt, invoke **DRAWGRAPH** by typing 'draw' with no argument. The drawing plate will be opened, followed by the menu dialog window. Position both windows at your convenience, and **DRAWGRAPH** is ready for drawing objects. To exit **DRAWGRAPH**, left-click the 'Exit' button.

Note: Before all this, you must do 'xinit' to initialize the X-Window environment if you have not previously. If your work station does not have **PROWINDOWS**, you must 'rxterm' (not rlogin) to a machine that does.

Creating Objects

The first step to draw an object is to select the pen size, unless a default size is desired. To select a pen size, click the pen size button using the mouse right button. Ten pen selections will be displayed. Select the desired pen size by clicking the mouse left button. Any object created after this selection will be drawn at the selected pen size. To return the default pen size, left-click the pen-size button. If the object is a line, path, or curve, and an arrow is desired to be on the line end, then 'Line ends' button should be selected before drawing the object.

Drawing Lines

Left-click the 'Line' button in **DRAWGRAPH** 'Create' menu to draw lines. A sub-menu for drawing lines as shown in Figure 9 will be displayed. This sub-menu is cancelled by left-clicking 'Done' button.

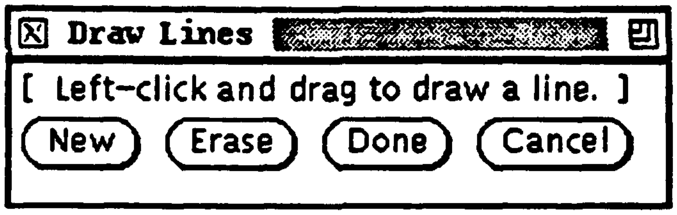


Figure 9: Sub-menu for drawing lines

To draw a new line, left-click 'New' button, and the button will appear 'greyed'. The cursor is now ready for drawing a line. Left-click again at a point in the drawing area, and drag the mouse while holding the button down to draw a line, release the mouse button to end the line. Arrows will be drawn at line ends as specified by the 'Line ends' selection. To draw another line, repeat left-clicking the 'New' button and left-click and drag the mouse. If after clicking the 'New' button drawing a line is not desired, the 'greyed' button can be released by clicking 'Cancel' button.

To erase a line, left-click the 'Erase' button, and then left-click the line to be erased. This process can be repeated as necessary. If for some reason 'Erase' is selected but decided not to be used, 'Cancel' button will release the 'greyed' 'Erase' button. After drawing lines, left-click 'Done' button, and the 'Draw Lines' sub-menu will disappear. Another menu selection then can be made.

Drawing Paths

Left-click the 'Path' button in **DRAWGRAPH** 'Create' menu to draw paths. A sub-menu for drawing paths like in Figure 10 will be displayed. This sub-menu is cancelled by clicking 'Done' button.

To draw a path, left-click 'Start' button, and the button will be 'greyed'. The cursor is now ready for drawing a path. Left-click and release the mouse at a starting point in the drawing area. Make another left-click to select next point for the path. A line connecting the two points will be drawn on the drawing plate. Another point can be selected, and another connecting line will be drawn. This procedure can be repeated as necessary. As in line drawing, arrows will be added at the path ends according to the 'Line ends' selection. To stop the path drawing click the 'Cancel' button from the sub-menu. To draw another path, left-click the 'Start' button again and repeat the process. The 'Erase' button works as with drawing straight lines.

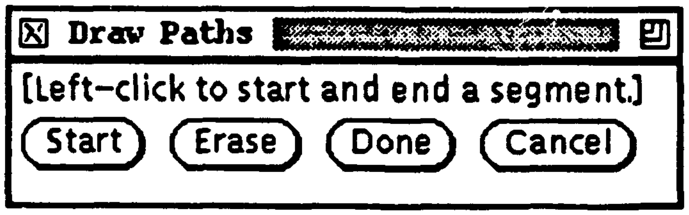


Figure 10: Sub-menu for drawing paths

Drawing Curves

A curve is a smoothed path. Left-click the 'Curve' button in **DRAWGRAPH** 'Create' menu to draw curves. A sub-menu for drawing curves similar to the sub-menu for drawing paths will be displayed. Options are like those for drawing paths.

Drawing Boxes

Left-click the 'Box' button in **DRAWGRAPH** 'Create' menu to draw boxes. A sub-menu for drawing boxes as shown in Figure 11 will be displayed. This sub-menu is cancelled by clicking 'Done' button.

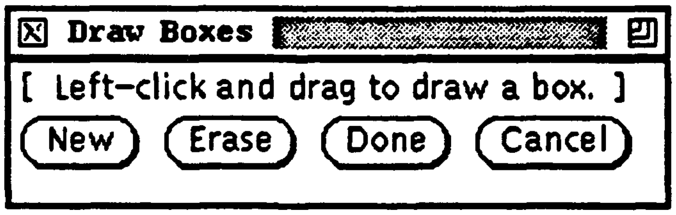


Figure 11: Sub-menu for drawing boxes

To draw boxes, left-click 'New' button. The button will be 'greyed', which means the cursor is ready for drawing a box. Left-click and drag the mouse to the lower right to draw a box, and release the mouse button to finalize the box. The box will be drawn with the starting point as the upper-left corner and the end point as the lower-right corner. Other options are similar to drawing previous objects.

Drawing Rounded-Boxes, Circles and Ellipses

A rounded box is a box with rounded corners. The procedure for drawing rounded-boxes, circles, and ellipses are the same as drawing boxes. Left-click the 'Rounded', 'Circle', or 'Ellipse' button in **DRAWGRAPH** menu to draw rounded-boxes, circles or ellipses. However, the program behaves differently as you move the mouse to the lower right corner and release it. For the rounded-boxes, the corner will be rounded after the left button is released. As for circles and ellipses, a circle or ellipse will be drawn inside the box after the button is released and the box will disappear. The sub-menu also looks alike.

Note: The circle will take the shortest of the box sides as its diameter.

Drawing Text-Boxes

A text-box is a box with text item centered inside the box. Drawing text-boxes consists of two steps: drawing the box and typing the text. The procedure for drawing the box is the same as drawing boxes. After the mouse left button is released, the text-typing sub-menu like in Figure 12 will be displayed. Left-click at the text area, a cursor will appears at the beginning of the line. Type in the desired text, and then left-click the 'OK' button. The text will be displayed centered inside the previously drawn box, and the 'Write Text' sub-menu will disappear.

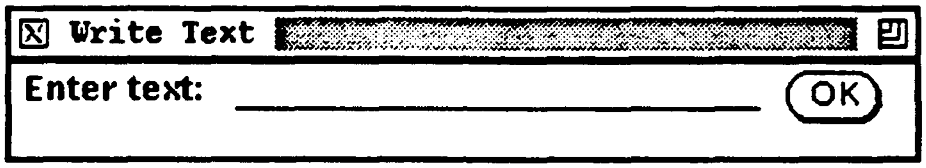


Figure 12: Sub-menu for writing text for text boxes

Note: Although the text area in the sub-menu appears limited, the actual area is not. It will scroll to the left as the text exceeds the blank line.

Shading

Objects can be filled with shades selected from the 'Fill' menu. To fill an object with a shade, first left-click the desired shade from the 'Fill' menu. The selected menu will be highlighted. Then bring the cursor to the object to be shaded, and left-click the object. The interior of the object will be filled with the selected shade. To change shade, change the shade selection by clicking the new selection, and proceed with the same procedure to shade the object. To cancel the fill, just select white fill and click the object. White fill is the default fill when the menu was created.

Warning: Prowindows doesn't quite completely fill some curved shapes.

Selecting Pen Size

The 'Pen size' menu is used to select the thickness of the lines drawn on the drawing surface. The menu is a cycle menu with selection of pen-size 1 to 10, indicating the line thickness in pixel. Pen size 1 is the default when the menu was created. Pen selection should be done before drawing the object, and subsequent objects will be drawn at this pen size selection.

Saving Objects

The 'Save' selection from the 'Operate' menu is used to save objects into bitmap files. To save objects, left-click the 'Save' selection from the **DRAWGRAPH** 'Operate' menu. A 'Clip to Save' sub-menu like in Figure 13 will be displayed.

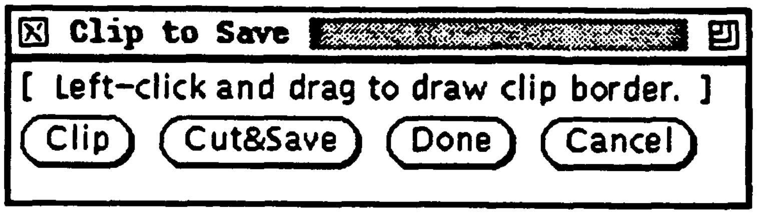


Figure 13: Sub-menu for saving objects

Left-click 'Clip' button from this sub-menu, then draw a border box around the area to be saved. This process is done exactly the same as drawing boxes in the 'Create' menu. It is important to draw the border as close as possible to the objects, so the space used to store the bitmap file will be minimum. After drawing the border, left-click 'Cut&Save' selection to continue the saving process.

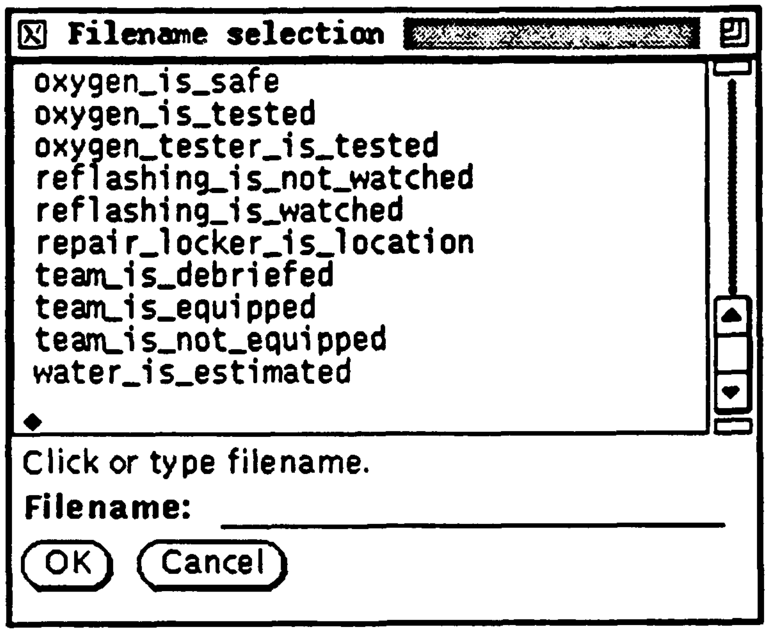
After clicking the 'Cut&Save' selection, a sub-menu for filename selection like in Figure 14 will be displayed. The sub-menu is a browser menu with selection of filenames taken from the teacher's tutoring program. This is the reason why the tutoring program should be loaded at the same time with **DRAWGRAPH** to make it available for this purpose. A filename can be selected from the browser by clicking a selection, or a new filename can be typed in on the space provided. The filename selected from the browser will be displayed on this space and can be edited as desired.

To actually save the enclosed objects click the sub-menu 'OK' button. To cancel saving objects click the 'Cancel' button from the filename selection sub-menu.

Note: The file names in the browser are sorted alphabetically for ease of selection.

Displaying a Saved File

A bitmap saved by **DRAWGRAPH** can be loaded and displayed at the same position as the position at its creation time. This is possible because the positions of the files are stored in a position file. This file is created and maintained throughout the session, and will be used by both the **DRAWGRAPH**, and the **MEGRAPH21** (the tutoring system) to position a picture on the screen. To display a file on the drawing plate click 'Display' selection from **DRAWGRAPH** menu. A filename selection sub-menu similar to the one used for saving objects will be displayed. The filename can be selected through the browser or defined by typing it. If the file is in the directory it will be loaded and displayed, otherwise the menu will be cancelled.



```
oxygen_is_safe
oxygen_is_tested
oxygen_tester_is_tested
reflashing_is_not_watched
reflashing_is_watched
repair_locker_is_location
team_is_debriefed
team_is_equipped
team_is_not_equipped
water_is_estimated
```

Figure 14: Sub-menu for selecting a filename

Note: A displayed object called using the above menu is different from a created object. A created object can be saved and manipulated like erased, moved, grouped, etc., but the displayed object cannot. A created object may overlap a displayed object, but can still be saved without being 'contaminated' by the displayed objects. This feature is an advantage for positioning a new object relative to previously defined ones.

Grouping and Ungrouping Objects

Primitive objects drawn on the drawing plate (lines, paths, curves, boxes, circles etc.) are individual objects. It can be moved, and rearranged to make a collection of objects. To make several primitive objects as one object so it can be moved together, those individual objects need to be grouped. The 'Group' selection of **DRAWGRAPH** 'Operate' menu is used to group selected primitive objects into one object. After grouping, the object can be manipulated as a unit. Operation like erase and move will work on the entire group member.

To group several objects into one group, select the 'Group' button from the 'Operate' menu. A 'Group Objects' sub-menu will be displayed. The next step is to select the member of the group. This is done by clicking the 'Select' button and left-click the selected object anywhere in the imaginary box surrounding it. The selected object will be displayed in 'greyed' form. Selection of objects is done one by one. After all member have been selected, click the 'Group' button. The selected members will be displayed normally, and they become one unit. The group sub-menu will disappear. To cancel operation any time before clicking the 'Group' button, left-click the 'Cancel' button.

The 'Ungroup' button is used to ungroup a previously grouped objects. Left-click the 'Ungroup' button and then select the grouped object. The selected object will now back as individual objects and can be manipulated individually.

Note: Group operation only works for individual primitive objects. To group a grouped objects, it must be ungroupped first into individual primitives.

Copying an Object

An object can be duplicated by left-clicking the 'Copy' button from the 'Operate' menu, followed by left-clicking the object to be duplicated. A duplicate object will be displayed five pixels away from the original. The duplicate object can be manipulated like the original object.

Moving Objects

Objects created in the drawing plate are movable, so they can be rearranged as desired. To move an object, middle-click the object anywhere inside an imaginary rectangle surrounding it, and drag the mouse while holding the middle button down to the desired position. Release the button to finalize the new position. The object is now drawn at the new position. When objects overlap each other sometime it is difficult to determine the correct rectangle area. Moving the overlapping objects around will solve this problem.

Final Note

The rectangle area surrounding an object will also prevent the starting of a drawing. This is because the drawing area recognizes the left-click inside the rectangle differently. To avoid this, start the drawing at some clearly free space, then move the object to the desired position. To finalize a drawing inside the rectangle area is all right.

APPENDIX B

MEGRAPH21 USER GUIDE

Introduction and Purpose

MEGRAPH21 is a domain independent intelligent computer-assisted instruction system that teaches sequential skills. The system uses objects either graphical or textual as user-computer interface. The tutoring is displayed in a main window, and the user selects an action from a list of actions displayed in a browser window using a mouse.

MEGRAPH21 is an augmentation of Professor Rowe's METUTOR21 [ROWE 90], and is written in PROWINDOWS, the object oriented graphic language extension of PROLOG. It compiles in PROWINDOWS under Unix X-Window environment. The users of MEGRAPH21 are teachers building a tutorial program and students learning the skill using the tutoring system. The users are assumed to be familiar with PROLOG.

MEGRAPH21 session window

MEGRAPH21 session window consists of one main window with a browser window on the right. A graphical display window displays the state of the tutoring graphically. Above the graphical display windows are the introduction display window, objectives display window, and current state window. Below the graphical display window are the action selection window and the tutoring window.

The introduction window gives the problem context. It introduces the tutor to the student as to give a general idea of what is happening. The objective window gives the objective definition, which is the final goal the student should achieve. The current state window shows the definition of the condition at this point of simulation. The action selection window displays the last operator selected by the student. The tutoring window displays whatever the tutor says or comments. This comments are made by the inference engine as the tutoring progresses.

Invoking MEGRAPH21

MEGRAPH21 requires that PROLOG and PROWINDOWS are properly installed, and can be called from MEGRAPH21 directory. To run MEGRAPH21 successfully the following files should reside and be loaded in the same directory:

1. MEGRAPH21.
2. COMMON, a utility file.
3. The teacher's tutorial program.

This is done by typing '[megraph21, common, program_name]' followed by <ENTER> at PROLOG prompt. If the graphical representations are already defined, the bitmap files and the text file that retains their position, posfile, must also be in the same directory These files are created by the teacher using **DRAWGRAPH**, a graphics tool (see **DRAWGRAPH** User Guide).

Note: Before al this, you must do 'xinit' to initialize the X- Window environment if you have not previously. If your work station dies not have **PROWINDOWS**, you must 'rxterm' (not rlogin) to a machine that does.

MEGRAPH21 Main Menu

After successful loading the tutoring system is started by typing 'go', and a main menu will be displayed. The main menu has three selection buttons: Go, Help and Exit.

Go: continue the tutoring system and start the learning process.
Help: show on-line help
Exit: back to **PROLOG** prompt

Note: No on-line help is provided at this moment. Pressing Help button will do nothing.

Learning Session

Clicking 'Go' from the main menu using the mouse left button will start the session and display the session window. Part of the session window that can be manipulated is the list of operators in the browser window. This window shows all possible operators in the tutorial program. To select an operator clicking the operator using the mouse left button. The selected operator will be greyed and the tutor states and graphics display will change according to changes caused by the applying of the operator. The selected operator will be displayed in the operator selection box under the graphical display.

Exiting the Session Window

When the session ends, the student exit the tutoring system by selecting Exit from the browser window. The session window will be closed, but the main menu is still displayed. Clicking 'Go' again will repeats the learning session.

Note: You can exit the learning session at any time.

Exiting MEGRAPH21

To exit **MEGRAPH21** completely, click 'Exit' button from the main menu. This will bring the **PROLOG** prompt back.

APPENDIX C

SOURCE CODE OF DRAWGRAPH

```
%=================================================================
% Program      : DrawGraph
% Purpose      : This graphics editor program is a tool for teachers to specify graphic
%                objects for their tutoring programs. Drawings are created using
%                primitives building-blocks such as lines, paths, boxes, circles etc. and
%                stored in a bitmap ile.
% Author       : Francius Suwono
% Date         : April 6, 1992
% Source       : Prolog Prowindows
% Notes        : This program should be loaded together with the following programs:
%                a. common, a utility file
%                b. the teacher's tutoring program
%                Also: posfile, a file that contains area information of objects should
%                be in the same directory.
% -----------------------------------------------------------------
% -----------------------------------------------------------------
% Main calling routine.
% draw/0 creates the main plate for drawing pictures, and makes the necessary
% preparation like loading information about previous picture position, calls
% the main menu and opens both drawing plate and the menu.
% -----------------------------------------------------------------
:- use_module(library(interpret_messages)).
:- use_module(library(dialog)).
:- use_module(library(messages)).
:- use_module(library(rename)).


draw :-
        not(object(@pic)) ->
        % Load picture position from posfile.
        (read_position,

        % Create main plate @pic
        clear(@pic),
        new(@pic, picture('Main plate', size(930,384))),
        send_list(@pic, [horizontal_scrollbar, vertical_scrollbar], off),

        clear(@bit),
        new(@bit, bitmap(930,384)),

        % display the main plate and the menu.
        send(@pic, open),
        draw_menu) | true.
```

```
% ------------------------------------------------------------------------
% draw_menu/0 creates menu to draw objects
% Object @maindraw is the main dialog window for drawing.
% It contains menus for:
%           - operate, for drawing manipulation
%           - create, to create primitive drawing
%           - fill, to fill area with pattern
%           - pen size, to select pen thickness
%           - line ends, to specify arrow for line's end
% ------------------------------------------------------------------------
draw_menu :-
          % the dialog window
          new(@maindraw, dialog('DrawGraph')),

          % menu for operate
          new(@operate, menu('Operate', choice, cascade(@maindraw, select, 0))),
          send_list(@operate, append, ['Group', 'Ungroup', 'Copy', 'Save',
          'Display', 'Erase', 'Clear', 'Redraw', 'Exit']),

          % menu for create
          new(@create, menu('Create', choice, cascade(@maindraw, select, 0))),
          send_list(@create, append, ['Line', 'Path', 'Curve', 'Box',
          'Rounded', 'Circle', 'Ellipse', 'Text Box']),

          % menu for fill.
          new(@fill, menu('Fill: ', choice, cascade(@maindraw, select, 0))),

          % make fill bitmaps for menu display
          make_fill,
          clear(@wht),
          clear(@gr1),
          clear(@gr2),
          clear(@gr3),
          clear(@gr4),
          clear(@blk),
          new(@wht, menu_item('White',0)),
          new(@gr1, menu_item('Grey1',0)),
          new(@gr2, menu_item('Grey2',0)),
          new(@gr3, menu_item('Grey3',0)),
          new(@gr4, menu_item('Grey4',0)),
          new(@blk, menu_item('Black',0)),
          send(@wht, image, @white),
          send(@gr1, image, @grey1),
          send(@gr2, image, @grey2),
          send(@gr3, image, @grey3),
          send(@gr4, image, @grey4),
          send(@blk, image, @black),
          send_list(@fill, append, [@wht, @gr1, @gr2, @gr3, @gr4, @blk]),
        % menu for pen size.
        new(@pen, menu('Pen size: ', cycle,cascade(@maindraw, select, 0))),
        send_list(@pen, append, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]),

        % menu for ends.
        new(@ends, menu('Line ends:', cycle, cascade(@maindraw, select, 0))),
        send_list(@ends, append, ['None', 'First Arrow','Second Arrow',
        'Both Arrows']),

        % arrange menu position.
        send(@maindraw, append, @operate),
        send(@create, right, @operate),
        send(@fill, right, @create),
        send(@pen, below, @fill),
        send(@ends, below, @pen),
        send(@maindraw, open).


% -------------------------------------------------------------------------
% make grey bitmap
% -------------------------------------------------------------------------
make_fill :-
        fill_white(@white),
        fill_grey1(@grey1),
        fill_grey2(@grey2),
        fill_grey3(@grey3),
        fill_grey4(@grey4),
        fill_black(@black).

fill_grey1(Bit) :-
        clear(Bit),
        new(Bit, bitmap(60,18)),
        new(Grey, bitmap(4,2)),
        send_list(Grey, set, [point(0,0), point(2,0)]),
        send(Bit, replicate, Grey).

fill_grey2(Bit) :-
        clear(Bit),
        new(Bit, bitmap(60,18)),
        new(Grey, bitmap(2,2)),
        send_list(Grey, set, [point(0,0), point(1,1)]),
        send(Bit, replicate, Grey).

fill_grey3(Bit) :-
        clear(Bit),
        new(Bit, bitmap(60,18)),
        new(Grey, bitmap(5,2)),
        send_list(Grey, set, [point(0,0), point(2,0), point(3,0), point(1,1),
        point(2,1), point(4,1)]),
        send(Bit, replicate, Grey).
```

```
fill_grey4(Bit) :-
            clear(Bit),
            new(Bit, bitmap(60,18)),
            new(Grey, bitmap(3,3)),
            send_list(Grey, set, [point(0,0), point(0,2), point(1,1), point(2,0),
            point(2,2)]),
            send(Bit, replicate, Grey).

fill_white(Bit) :-
            clear(Bit),
            new(Bit, bitmap(60,18)),
            send(Bit, clear).

fill_black(Bit) :-
            clear(Bit),
            new(Bit, bitmap(60,18)),
            send(Bit, clear),
            send(Bit, invert).


% ----------------------------------------------------------------------------
% selections for operate.
% ----------------------------------------------------------------------------
select(Menu,'Group') :-
            not(object(@exit)),
            group_object.

select(Menu,'Ungroup') :-
            not(object(@exit)),
            ungroup_object.

select(Menu,'Copy') :-
            not(object(@exit)),
            copy_object.

select(Menu, 'Save') :-
            not(object(@exit)),
            save_object.

select(Menu,'Display') :-
            not(object(@exit)),
            display_object.

select(Menu,'Erase') :-
            not(object(@exit)),
            erase_object.

select(Menu,'Clear') :-
            not(object(@exit)),
            send(@pic, clear).
```

```
select(Menu,'Redraw') :-
        not(object(@exit)),
        send(@pic, redraw).

select(Menu,'Exit') :-
        not(object(@exit)),
        exit_draw.


% -----------------------------------------------------------------------
% selections to create objects
% -----------------------------------------------------------------------
select(Menu,'Line') :-
        not(object(@exit)),
        create_line.

select(Menu,'Path') :-
        not(object(@exit)),
        create_path.

select(Menu,'Curve') :-
        not(object(@exit)),
        create_curve.

select(Menu,'Box') :-
        not(object(@exit)),
        create_box.
select(Menu,'Rounded') :-
        not(object(@exit)),
        create_rounded.

select(Menu,'Circle') :-
        not(object(@exit)),
        create_circle.

select(Menu,'Ellipse') :-
        not(object(@exit)),
        create_ellipse.

select(Menu,'Text Box') :-
        not(object(@exit)),
        create_textbox.


% -----------------------------------------------------------------------
% selection for pen
% -----------------------------------------------------------------------
pen(X, Pen) :-
        get(@pen, selection, X),
        atom_chars(X, [N]),
        Pen is N - 48.
```

```
% ---------------------------------------------------------------------------
% selection for fill.
% ---------------------------------------------------------------------------
select(Menu,'Grey1') :-
            fill_object(@pic, @grey1).
select(Menu,'Grey2') :-
            fill_object(@pic, @grey2).

select(Menu,'Grey3') :-
            fill_object(@pic, @grey3).

select(Menu,'Grey4') :-
            fill_object(@pic, @grey4).

select(Menu,'White') :-
            fill_object(@pic, @white).

select(Menu,'Black') :-
            fill_object(@pic, @black).
fill_object(Picture, Fill) :-
            send(Picture, clicked, cascade(Picture, fill, Fill)).

fill(Picture, Fill) :-
            get_ref(Picture, current, Fig),
            get_ref(Fig, graphicals, Ch),
            get(Ch, list_refs, [Graphical|_]),
            send(Graphical, fill, Fill).


% ---------------------------------------------------------------------------
% selections for ends.
% ---------------------------------------------------------------------------
arrows(Line, Ends):-
            get(@ends, selection, Ends),
            put_arrows(Line, Ends).

put_arrows(Line, 'First Arrow') :-
            send(Line, arrows, first).

put_arrows(Line, 'Second Arrow') :-
            send(Line, arrows, second).

put_arrows(Line, 'Both Arrows') :-
            send(Line, arrows, both).

put_arrows(Line, _).
```

```
% ------------------------------------------------------------------------
% routines for creating objects
% ------------------------------------------------------------------------

% ------------------------------------------------------------------------
% create lines
% ------------------------------------------------------------------------
create_line :-
        not(object(@draw_palette)) ->
        (new_dialog(@draw_palette, 'Draw Lines', draw_line),
        send(@draw_palette, open)) | true.

% callbacks
line_new(Picture, _) :-
        retractall(create_flag),
        send(Picture, left_down, cascade(Picture, start_line, 0)),
        process_callbacks(retract(create_flag)).

start_line(Picture, Pos) :-
        get(@pic, last_click, Point),
        Point = point(X,Y),
        new(Line, line(X,Y,X,Y)),
        pen(Size, Pen),
        arrows(Line, Ends),
        send(Line, pen, Pen),
        send(Line, start, Pos),
        send(@pic, left_drag, message(Line, end, 0)),
        new(Fig, figure),
        send(Fig, append, Line),
        send(@pic, display, Fig),
        send(@pic, left_up, cascade(@pic, graph_cancel, 0)).

% generic callbacks for erase, cancel and done
graph_erase(Picture,_) :-
        send(Picture, clicked, cascade(Picture, erase, 0)),
        process_callbacks(retract(create_flag)).

erase(Picture, _) :-
        get_ref(Picture, current, Fig),
        get_ref(Fig, graphicals, Ch),
        get(Ch, list_refs, [Graphical|_]),
        clear(Fig),
        create_cancel(Picture, _).

graph_done(Picture, Dialog) :-
        create_cancel(Picture, _),
        clear(Dialog).

graph_cancel(Picture, Dialog) :-
        create_cancel(Picture, _).
create_cancel(Picture, _) :-
        send(Picture, clicked, 0),
        send(Picture, left_up, 0),
        send(Picture, left_down, 0),
        send(Picture, left_drag, 0),
        uniqueassert(create_flag).


% dialog declarations
draw_line(_, label('[ Left-click and drag to draw a line. ]',0), below, []).
draw_line(_, button('New', cascade(@pic, line_new, 0)), below, []).
draw_line(_, button('Erase', cascade(@pic, graph_erase, 0)), right, []).
draw_line(_, button('Done', cascade(@pic, graph_done, @draw_palette)), right, []).
draw_line(_, button('Cancel', cascade(@pic, graph_cancel, @draw_palette)), right, []).


% ---------------------------------------------------------------------------------
% create paths
% ---------------------------------------------------------------------------------
create_path :-
        not(object(@draw_palette)) ->
        (new_dialog(@draw_palette, 'Draw Paths', draw_path),
        send(@draw_palette, open)) | true.

% callbacks
path_start(Picture,_) :-
        retractall(create_flag),
        send(Picture, left_up, cascade(Picture, path, 0)),
        process_callbacks(retract(create_flag)).

path(Picture, Pos) :-
        new(Fig,figure),
        new(Path, path),
        pen(Size, Pen),
        send(Path, pen, Pen),
        send(Path, append, Pos),
        send(Fig, append, Path),
        send(Picture, display, Fig),
        send(Picture, left_up, message(Path, append,0)).


% dialog declarations
draw_path(_, label('[Left-click to start and end a segment.]',0), below, []).
draw_path(_, button('Start', cascade(@pic, path_start, 0)), below, []).
draw_path(_, button('Erase', cascade(@pic, graph_erase, 0)), right, []).
draw_path(_, button('Done', cascade(@pic, graph_done, @draw_palette)), right, []).
draw_path(_, button('Cancel', cascade(@pic, graph_cancel, @draw_palette)), right, []).
```

```
% ----------------------------------------------------------------------
% create curves
% ----------------------------------------------------------------------
create_curve :-
        not(object(@draw_palette)) ->
        (new_dialog(@draw_palette, 'Draw Curves', draw_curve),
        send(@draw_palette, open)) | true.


% callbacks
curve_start(Picture, _) :-
        retractall(create_flag),
        send(Picture, left_up, cascade(Picture, curve, 0)),
        process_callbacks(retract(create_flag)).
curve(Picture, Pos) :-
        new(Fig,figure),
        new(Curve, path),
        pen(Size, Pen),
        send(Curve, pen, Pen),
        send(Curve, append, Pos),
        send(Curve, smooth, on),
        send(Curve, intervals, 10),
        send(Fig, append, Curve),
        send(Picture, display, Fig),
        send(Picture, left_up, message(Curve, append,0)).

% dialog declarations
draw_curve(_, label('[Left-click to start and end a segment.]',0), below, []).
draw_curve(_, button('Start', cascade(@pic, curve_start, 0)), below, []).
draw_curve(_, button('Erase', cascade(@pic, graph_erase, 0)), right, []).
draw_curve(_, button('Done', cascade(@pic, graph_done, @draw_palette)), right, []).
draw_curve(_, button('Cancel', cascade(@pic, graph_cancel, @draw_palette)), right, []).

% ----------------------------------------------------------------------
% create boxes
% ----------------------------------------------------------------------
create_box :-
        not(object(@draw_palette)) ->
        (new_dialog(@draw_palette, 'Draw Boxes', draw_box),
        send(@draw_palette, open)) | true.

% callbacks
box_new(Picture, _) :-
        retractall(create_flag),
        send(Picture, left_down, cascade(Picture, start_box, 0)),
        process_callbacks(retract(create_flag)).

start_box(Picture, Pos) :-
        new(Fig,figure),
            new(Box, box),
            pen(Size, Pen),
            send(Box, pen, Pen),
            send(Box, position, Pos),
            send(Picture, left_drag, message(Box, corner, 0)),
            send(Fig, append, Box),
            send(Picture, display, Fig),
            send(Picture, left_up, cascade(Picture, refresh, 0)).

refresh(Picture, _) :-
            send(Picture, redraw),
            create_cancel(Picture, _).


% dialog declarations
draw_box(_, label('[  Left-click and drag to draw a box.  ]',0), below, []).
draw_box(_, button('New', cascade(@pic, box_new, 0)), below, []).
draw_box(_, button('Erase', cascade(@pic, graph_erase, 0)), right, []).
draw_box(_, button('Done', cascade(@pic, graph_done, @draw_palette)), right, []).
draw_box(_, button('Cancel', cascade(@pic, graph_cancel, @draw_palette)), right, []).


% ----------------------------------------------------------------------------------
% create rounded boxes
% ----------------------------------------------------------------------------------
create_rounded :-
            not(object(@draw_palette)) ->
            (new_dialog(@draw_palette, 'Draw Rounded', draw_rounded),
            send(@draw_palette, open)) | true.

% callbacks
rounded_new(Picture, _) :-
            retractall(create_flag),
            send(Picture, left_down, cascade(Picture, start_rounded, 0)),
            process_callbacks(retract(create_flag)).

start_rounded(Picture, Pos) :-
            new(Fig,figure),
            new(Rounded, box),
            pen(Size, Pen),
            send(Rounded, pen, Pen),
            send(Rounded, position, Pos),
            send(Picture, left_drag, message(Rounded, corner, 0)),
            send(Fig, append, Rounded),
            send(Picture, display, Fig),
            send(Picture, left_up, cascade(Picture, radius, Rounded)).

radius(Picture, Rounded) :-
            send(Picture, redraw),
            send(Rounded, radius, 8),
            create_cancel(Picture, _).
```

```
% dialog declarations
draw_rounded(_, label('[Left-click and drag to draw a rounded box.]',0), below, []).
draw_rounded(_, button('New', cascade(@pic, rounded_new, 0)), below, []).
draw_rounded(_, button('Erase', cascade(@pic, graph_erase, 0)), right, []).
draw_rounded(_, button('Done', cascade(@pic, graph_done, @draw_palette)),
        right, []).
draw_rounded(_, button('Cancel', cascade(@pic, graph_cancel, @draw_palette)),
        right, []).


% ------------------------------------------------------------------------
% create circles
% ------------------------------------------------------------------------
create_circle :-
        not(object(@draw_palette)) ->
        (new_dialog(@draw_palette, 'Draw Circle', draw_circle),
        send(@draw_palette, open)) | true.

% callbacks
circle_new(Picture, _) :-
        retractall(create_flag),
        send(Picture, left_down, cascade(Picture, start_circle, 0)),
        process_callbacks(retract(create_flag)).

start_circle(Picture, Pos) :-
        clear(@b),
        clear(@f),
        new(@b, box),
        new(@f, figure),
        send(@b, position, Pos),
        send(Picture, left_drag, message(@b, corner, 0)),
        send(@f, append, @b),
        send(@f, greyed, on),
        send(Picture, display, @f),
        send(Picture, left_up, cascade(Picture, draw_circle, 0)).

draw_circle(Picture, _) :-
        send(Picture, erase, @f),
        graph_cancel(Picture, _),
        get(@b, area, area(X,Y,W,H)),
        min(W, H, D),

        % draw the circle
        new(Fig,figure),
        new(Circle, circle),
        send(Circle, area, area(X, Y, D, D)),
        pen(Size, Pen),
        send(Circle, pen, Pen),
        send(Fig, append, Circle),
        send(Picture, display, Fig),
        clear(@b),
        clear(@f),
        send(Picture, redraw).

% dialog declarations
draw_circle(_, label('[Left-click and drag to draw a circle.]',0), below, []).
draw_circle(_, button('New', cascade(@pic, circle_new, 0)), below, []).
draw_circle(_, button('Erase', cascade(@pic, graph_erase, 0)), right, []).
draw_circle(_, button('Done', cascade(@pic, graph_done, @draw_palette)), right, []).
draw_circle(_, button('Cancel', cascade(@pic, graph_cancel, @draw_palette)),
        right, []).


% ----------------------------------------------------------------------
% create ellipses
% ----------------------------------------------------------------------
create_ellipse :-
        not(object(@draw_palette)) ->
        (new_dialog(@draw_palette, 'Draw Ellipse', draw_ellipse),
        send(@draw_palette, open)) | true.

% callbacks
ellipse_new(Picture, _) :-
        retractall(create_flag),
        send(Picture, left_down, cascade(Picture, start_ellipse, 0)),
        process_callbacks(retract(create_flag)).

start_ellipse(Picture, Pos) :-
        clear(@b),
        clear(@f),
        new(@b, box),
        new(@f, figure),
        send(@b, position, Pos),
        send(Picture, left_drag, message(@b, corner, 0)),
        send(@f, append, @b),
        send(@f, greyed, on),
        send(Picture, display, @f),
        send(Picture, left_up, cascade(Picture, draw_ellipse, 0)).

draw_ellipse(Picture, _) :-
        graph_cancel(Picture, _),
        get(@b, area, area(X,Y,W,H)),

        % draw the ellipse
        new(Fig,figure),
        new(Ellipse, ellipse),
        send(Ellipse, area, area(X, Y, W, H)),
        pen(Size, Pen),
        send(Ellipse, pen, Pen),
        send(Fig, append, Ellipse),
        send(Picture, display, Fig),
        clear(@b),
          clear(@f),
          send(Picture, redraw).

% dialog declarations
draw_ellipse(_, label('[Left-click and drag to draw an ellipse.]',0), below, []).
draw_ellipse(_, button('New', cascade(@pic, ellipse_new, 0)), below, []).
draw_ellipse(_, button('Erase', cascade(@pic, graph_erase, 0)), right, []).
draw_ellipse(_, button('Done', cascade(@pic, graph_done, @draw_palette)), right, []).
draw_ellipse(_, button('Cancel', cascade(@pic, graph_cancel, @draw_palette)),
          right, []).

% ----------------------------------------------------------------------------
% create text blocks
% ----------------------------------------------------------------------------
create_textbox :-
          not(object(@draw_palette)) ->
          (new_dialog(@draw_palette, 'Draw Textbox', draw_textbox),
          send(@draw_palette, open)) | true.

% callbacks
textbox_new(Picture, _) :-
          retractall(create_flag),
          send(Picture, left_down, cascade(Picture, start_textbox, 0)),
          process_callbacks(retract(create_flag)).

start_textbox(Picture, Pos) :-
          clear(@box),
          clear(@tb),
          clear(@fig),
          new(@fig, figure),
          new(@box, box),
          new(@tb, text_block),
          send(@box, position, Pos),
          send(@fig, append, @box),
          send(@fig, append, @tb),
          send(Picture, display, @fig),
          send(Picture, left_drag, message(@box, corner, 0)),
          send(Picture, left_up, cascade(Picture, create_text, 0)),
          clear(@draw_text).

create_text(Picture, _) :-
          new_dialog(@draw_text, 'Write Text', draw_text),
          send(@draw_text, open).

textbox_ok(Picture, _) :-
          get(@box, area, Area),
          send(@tb, area, Area),
          send(@tb, format, center),
          get(@ti, selection , Text),
          send(@tb, string, Text),
        get_ref(@box, duplicate, Box),
        new(Tb, text_block(Text, Area, center)),
        duplicate_textbox(Tb, Box),
        clear(@draw_text),
        create_cancel(Picture, _).

duplicate_textbox(Tb, Box) :-
        new(Fig, figure),
        send_list(Fig, append, [Tb, Box]),
        get(Tb, area, Area),
        send(@pic, display, Fig),
        clear(@fig).
% dialog declarations
draw_textbox(_, label('[Left-click and drag to draw the box.]',0), below, []).
draw_textbox(_, button('New', cascade(@pic, textbox_new, 0)), below, []).
draw_textbox(_, button('Erase', cascade(@pic, graph_erase, 0)), right, []).
draw_textbox(_, button('Done', cascade(@pic, graph_done, @draw_palette)), right, []).
draw_textbox(_, button('Cancel', cascade(@pic, graph_cancel, @draw_palette)), right, []).

draw_text(@ti, text_item('Enter text: ', '', 0), below, []).
draw_text(_, button('OK', cascade(@pic, textbox_ok, 0)), right, []).


% -----------------------------------------------------------------------
% subroutines for operates
% -----------------------------------------------------------------------
% -----------------------------------------------------------------------
% subroutines for groupping objects
% -----------------------------------------------------------------------
group_object :-
        demolish(@oldfig_holder),
        demolish(@group_holder),
        demolish(@pos_holder),
        (not(object(@draw_pallete)) ->
        (new(@group_holder, chain),
        new(@oldfig_holder, chain),
        new(@pos_holder, chain),
        new_dialog(@draw_pallete, 'Group Objects', group),
        send(@draw_pallete, open))) | true.

% callbacks
group_select(Picture, _) :-
        retractall(operate_flag),
        send(Picture, clicked, cascade(Picture, select_group, 0)),
        process_callbacks(retract(operate_flag)).


select_group(Picture, _) :-

        % mark figure with greyed display
        get_ref(Picture, current, Fig),
                send(Fig, greyed, on),

                % get figure position
                get_ref(Picture, current, Fig),
                get(Fig, position, Pos),
                send(@pos_holder, append, Pos),

                % get the graphical
                get_ref(Fig, graphicals, Ch),
                get(Ch, list_refs, [Graphical|_]),
                send(@group_holder, append, Graphical),

                % append old figure to figure holder
                send(@oldfig_holder, append, Fig),
                send(Picture, clicked, 0),
                uniqueassert(operate_flag).


group(Picture, _) :-

                % erase old figure from display
                get(@oldfig_holder, list_refs, List),
                send_list(List, erase),

                % make position correction
                get(@group_holder, list_refs, Grouplist),
                get(@pos_holder, list, Poslist),
                correct_position(Grouplist, Poslist),

                % make a new figure for the group
                new(Fig, figure),
                send_list(Fig, append, Grouplist),
                send(Picture, display, Fig),

                % clean-up
                operate_cancel(Picture, _),
                send_list(@group_holder, delete, Grouplist),
                demolish(@oldfig_holder),
                demolish(@group_holder),
                demolish(@pos_holder),
                operate_cancel(Picture, _),
                clear(@draw_pallete).

correct_position([], []).
correct_position([G|G1], [P|P1]) :-
                send(G, position, P),
                correct_position(G1,P1).

group_cancel(Picture, _) :-
%               greyed_off(@oldfig_holder),
                demolish(@oldfig_holder),
          demolish(@group_holder),
          demolish(@pos_holder),
          clear(@draw_pallete).
          operate_cancel(Picture, _).

greyed_off([]).

greyed_off([F,F1]) :-
          send(F, greyed, off),
          greyed(F1).

operate_cancel(Picture, _) :-
          send(Picture, clicked, 0),
          send(Picture, left_up, 0),
          send(Picture, left_down, 0),
          send(Picture, left_drag, 0),
          uniqueassert(operate_flag).

% dialog declarations
group(_, label('[ Select drawing by left-cliking. ]',0), below, []).
group(_, button('Select', cascade(@pic, group_select, 0)), below, []).
group(_, button('Group', cascade(@pic, group, 0)), right, []).
group(_, button('Cancel', cascade(@pic, group_cancel, 0)),
          right, []).


clear_chain :-
          demolish(@oldfig_holder),
          demolish(@group_holder),
          demolish(@pos_holder).


% ---------------------------------------------------------------------------
% subroutines for ungroupping objects
% ---------------------------------------------------------------------------
ungroup_object :-
          not(object(@draw_pallete)) ->
          (new_dialog(@draw_pallete, 'UnGroup Objects', un_group),
          send(@draw_pallete, open)) | true.

% callbacks
ungroup_select(Picture, _) :-
          retractall(operate_flag),
          send(Picture, clicked, cascade(Picture, select_ungroup, 0)),
          process_callbacks(retract(operate_flag)).

select_ungroup(Picture, _) :-

          % greyed the selected figure
          get_ref(Picture, current, Fig),
          send(Fig, greyed, on),
            send(Picture, clicked, 0),
            uniqueassert(operate_flag).

un_group(Picture, _) :-
            get_ref(Picture, current, Fig),
            get(Fig, reference_point, point(X,Y)),

            % get the graphical
            get_ref(Fig, graphicals, Ch),
            get(Ch, list_refs, Graphicals),
            % ungroup it
            separate(Picture, Fig, Graphicals, X, Y),
            send(Fig, erase),
            send(Picture, redraw),
            operate_cancel(Picture, _),
            clear(@draw_pallete).


separate(P, F, [], _, _).
separate(P, F, [G|G1], X, Y) :-
            send(F, delete, G),
            get(G, position, point(X1, Y1)),
            X2 is X1 + X,
            Y2 is Y1 + Y,
            send(G, position, point(X2, Y2)),
            new(Fig, figure),
            send(Fig, append, G),
            send(P, display, Fig),
            separate(P, F, G1, X, Y).

ungroup_cancel(Picture, _) :-
            operate_cancel(Picture, _),
            clear(@draw_pallete).

% dialog declarations
un_group(_, label('[ Select drawing by left-cliking. ]',0), below, []).
un_group(_, button('Select', cascade(@pic, ungroup_select, 0)), below, []).
un_group(_, button('Un-group', cascade(@pic, un_group, 0)), right, []).
un_group(_, button('Cancel', cascade(@pic, ungroup_cancel, 0)),
            right, []).

% ---------------------------------------------------------------------------
% subroutines for copying object
% ---------------------------------------------------------------------------
copy_object :-
            send(@pic, clicked, cascade(@pic, copy, 0)).

copy(Picture, _) :-
            get_ref(Picture, current, Fig),
            get(Fig, reference_point, point(X,Y)),
            % get the graphical
            get_ref(Fig, graphicals, Ch),
            get(Ch, list_refs, Graphicals),
            % copy each member
            new(Fig1, figure),
            duplicate(Picture, Fig1, Graphicals, X, Y),
            send(Picture, display, Fig1),
            send(Picture, clicked, 0).

duplicate(P, F, [], _, _).

duplicate(P, F, [G|Rest], X, Y) :-
            get(G, area, area(X1, Y1, W1, D1)),
            X2 is X1 + X + 5,
            Y2 is Y1 + Y + 5,
            get_ref(G, duplicate, G1),
            send(G1, area, area(X2, Y2, W1, D1)),
            send(F, append, G1),
            duplicate(P, F, Rest, X, Y).


% ----------------------------------------------------------------------------
% subroutines for erasing object
% ----------------------------------------------------------------------------
erase_object :-
            send(@pic, clicked, cascade(@pic, erase1, 0)).

erase1(Picture, _) :-
            get_ref(Picture, current, Fig),
            get_ref(Fig, graphicals, Ch),
            get(Ch, list_refs, [Graphical|_]),
            clear(Fig),
            send(Picture, clicked, 0).


% ----------------------------------------------------------------------------
% subroutines for saving object
% ----------------------------------------------------------------------------
save_object :-
            clear(@clipped),
            (not(object(@draw_palette)) ->
            (new_dialog(@draw_palette, 'Clip to Save', draw_clip),
            send(@draw_palette, open)) | true).

% callbacks
clip(Picture, _) :-
            retractall(operate_flag),
            send(Picture, left_down, cascade(Picture, start_clip, 0)),
            process_callbacks(retract(operate_flag)).
```

```
start_clip(Picture, Pos) :-
        clear(@clipper),
        clear(@box),
        new(@clipper,figure),
        new(@box, box),
        send(@box, position, Pos),
        send(Picture, left_drag, message(@box, corner, 0)),
        send(@clipper, append, @box),
        send(Picture, display, @clipper),
        assert(operate_flag).

clip_cancel(Picture, _) :-
        uniqueassert(operate_flag),
        clear(@clipper).

cut_save(Picture, _) :-
        get_ref(Picture, figures, Fig_Chain),
        get(Fig_Chain, list_refs, Figures),
        draw_figure(@bit, Figures),
        get(@box, area, area(X, Y, W, H)),
        send(@bit, clip_area, area(X, Y, W, H)),
        get_ref(@bit, clip, Clipped_Bit),
        rename(Clipped_Bit, @clipped),
        select_filename(@save).

draw_figure(Bitmap, [@clipper]).
draw_figure(Bitmap, [Fig|Fig_Rest]) :-
        get(Fig, reference_point, point(X, Y)),
        get_ref(Fig, graphicals, Ch),
        get(Ch, list_refs, Graph_List),
        draw_graphicals(Bitmap, Graph_List, X, Y),
        draw_figure(Bitmap, Fig_Rest).

draw_graphicals(Bitmap,[], _, _).
draw_graphicals(Bitmap, [Graph|Graph_Rest], X, Y) :-
        get(Graph, position, point(X1, Y1)),
        X2 is X1 + X,
        Y2 is Y1 + Y,
        send(Graph, position, point(X2, Y2)),
        send(Bitmap, draw_in, Graph),
        send(Graph, position, point(X1, Y1)),
        draw_graphicals(Bitmap, Graph_Rest, X, Y).

save_done(Picture, Dialog) :-
        send(@bit, clear),
        clear(@clipper),
        graph_done(Picture, Dialog).
```

```
% callbacks for saving to filename
% if OK use the string text as filename to store the drawing
% save drawing to Filename, and assert the drawing position as fact

pressed(@save, 'OK') :-
        get(@ti, selection, Filename),
        send(@clipped, save, Filename),
        % assert the position reference
        get(@bit, clip_area, area(X, Y, W, H)),
        P =.. [area, Filename, X, Y, W, H],
        uniqueassert_area(P),
        write_position,
        clear(@save),
        clear(@clipper),
        clear(@clipped).

pressed(@save, 'Cancel') :-
        clear(@clipper),
        clear(@save).

% dialog declarations for clipping
draw_clip(_, label('[  Left-click and drag to draw clip border.  ]',0), below, []).
draw_clip(_, button('Clip', cascade(@pic, clip, 0)), below, []).
draw_clip(_, button('Cut&Save', cascade(@pic, cut_save, 0)), right, []).
draw_clip(_, button('Done', cascade(@pic, save_done, @draw_palette)), right, []).
draw_clip(_, button('Cancel', cascade(@pic, clip_cancel, 0)), right, []).


% ------------------------------------------------------------------------
% subroutines for displaying objects
% ------------------------------------------------------------------------
display_object :-
        not(object(@draw_palette)) ->
        (select_filename(@show)) | true.

pressed(@show, 'OK') :-
        get(@ti, selection, Filename),
        draw(Filename),
        clear(@show).

pressed(@show, 'Cancel') :-
        clear(@show).

draw(Filename) :-
        area(Filename, X, Y, W, H),
        new(Temp, bitmap(W, H)),
        send(Temp, load, Filename),
        send(Temp, position, point(X,Y)),
        send(@pic, display, Temp).
```

```
% ---------------------------------------------------------------------------
% exit drawgraph
% ---------------------------------------------------------------------------
exit_draw :-
            not(object(@draw_palette)) ->
            (new_dialog(@exit, 'Prompter', exit),
            send(@exit, open)) | true.
% dialog declarations for exiting the program
exit(_, label('Do you really want to exit DrawGraph?'), _, []).
exit(_, button('OK', pressed), below, []).
exit(_, button('Cancel', pressed), right, []).

% exit the program and clear all objects.
pressed(@exit, 'OK') :-

            % save all pictures' position
            write_position,
            clear_draw,
            clear(@exit).

% ---------------------------------------------------------------------------
% common subroutine used for operate
% ---------------------------------------------------------------------------
% cancel a dialog.
pressed(Dialog, 'Cancel') :-
            clear(Dialog).

% clear everything before exit
clear_draw :-
            clear(@maindraw),
            clear(@pic),
            clear(@operate),
            clear(@create),
            clear(@fill),
            clear(@pen),
            clear(@ends),
            clear(@exit),
            clear(@bit),
            clear(@ti).

% Eof DrawGraph
```

APPENDIX D

SOURCE CODE OF MEGRAPH21

```
%=======================================================================
% Program       : MeGraph21
% Purpose       : Problem-independent code for 'means-ends tutoring':
%                 tutoring for learning of sequences modelable by means-ends analysis.
% Version       : This is a Prowindows version of Prof. Rowe's METUTOR21
%                 intended to run on Quintus Prolog 3.0. Reorganized and
%                 augmented for graphics user interface.
% Author        : Prof. Neil C. Rowe
% Modified by   : Francius Suwono
% Date of mod.  : May 1, 1992
% Note          :
%                 For an application, you must define:
%
%                 (1) recommended(<difference>,<operator>)
%                 --recommendation conditions
%                 (2) precondition(<operator>,<factlist>) or
%                 precondition(<operator>,<conditionlist>,<factlist>) or
%                 precondition(<operator>,<conditionlist>,<factlist>,<msg>)
%                 --gives facts required by operator;
%                 3-arg. form requires additional facts true
%                 4-arg. form also prints message when precondition applied
%                 (3) deletepostcondition(<operator>,<factlist>) or
%                 deletepostcondition(<operator>,<conditionlist>,<factlist>)
%                 deletepostcondition(<operator>,<conditionlist>,<factlist>,<msg>)
%                 --gives facts deleted by op.;
%                 3-arg. form requires additional facts true;
%                 4-arg. form also prints message when applied
%                 (4) addpostcondition(<operator>,<factlist>) or
%                 addpostcondition(<operator>,<conditionlist>,<factlist>)
%                 addpostcondition(<operator>,<conditionlist>,<factlist>,<msg>)
%                 --gives facts added by op.;
%                 3-arg. form requires additional facts true
%                 4-arg. form also prints message when applied
%
%                 Some optional definitions you may include:
%
%                 (5) randsubst(<op.>,[<substlist1>,<substlist2>,...])
%                 --gives random-substitution triples or quadruples, each in the form:
%                 [<initial-fact>,<ending-fact>,<transition-prob.>,<message to user>
%                 Note: first and second arguments can be the word 'none';
%                 fourth argument is optional.
%                 (6) nopref(<operator1>,<operator2>)
%                 --if the order (priority) of two operators in the 'recommended'
%                 rules was arbitrary, include this fact.
%                 (7) intro(<text>) --introductory info for student
%              (8) debugflag --if asserted, debugging info printed re means-ends anal.
%              (9) studentflag -- if asserted, does not check for teacher errors
%
%              Also: as this tutor works, it asserts "student_error" facts that
%              log all student mistakes. To see, type "listing(student_error)"
%              to the top level of Prolog.
%              To use, there are two entries: "initialize_tutor" and "tutor(Op)".
% ------------------------------------------------------------------------


:- dynamic session_num/1, error_num/1, student_error/6,
        top_goal/1, top_solution/1, readbuff/1, op_list/1,
        mainline_states/4, cached/4, cached_disaster_op/2, current_state/1,
        tutor_says_done/1.
:- no_style_check(single_var), unknown(A,fail), load_files(library(random)).

session_num(0).
runtime_entry(go).
studentflag.

:- use_module(library(interpret_messages)).
:- use_module(library(dialog)).
:- use_module(library(messages)).


% ------------------------------------------------------------------------
% Creates the main menu and the main display window.
% Main menu window, has three selections :
%       Go : continue the tutoring system
%       Help : display help.
%       Exit: return to system
% ------------------------------------------------------------------------
go :-
        main_menu.

main_menu :-
        % open the main-menu dialog only if it is not opened yet
        not(object(@main_menu)) ->
        (new_dialog(@main_menu, cais, main_menu),
        send(@main_menu, size, size(200,60)),
        send(@main_menu, open)) |
        % otherwise do nothing
        true.

% callbacks for main menu of tutoring system
pick(@main_menu, 'Help').

pick(@main_menu, 'Exit') :-
        clear(@main_menu),
        clear_all.
```

```
pick(@main_menu, 'Go..') :-
            not(object(@main))->
            initialize_tutor,
            create_graph,
            run.

% dialog declarations for tutoring system main menu
% dialog declarations for tutoring system main menu
main_menu(_, label('Main Menu'), _, []).
main_menu(_, button('Go..', pick), below, []).
main_menu(_, button('Help', pick), right, []).
main_menu(_, button('Exit', pick), right, []).


% ------------------------------------------------------------------------
% Create main window for the tutoring system
% ------------------------------------------------------------------------

create_graph :-
            % load picture position from the file posfile
            read_position,

            % create main plate
            new(@main, picture('Computer Assisted Instructions System')),
            send(@main, size, size(930,850)),
            send_list(@main, [horizontal_scrollbar, vertical_scrollbar], off),

            % create operator browser
            new(@oplist, browser('Computer Assisted Intructions System')),
            send(@oplist, size, size(200,850)),

            % arrange and display position of menu boxes
            send(@oplist, right, @main),
            send(@main, open),
            % create fonts for text
            new(@font, font(gallant, bold, 14, 0)),
            new(@font2, font(gallant, bold, 18, 0)),

            % create and display intro heading and introduction
            display_intro,

            % create and display objectives heading
            new(@obj_heading, text_block('Your objectives:', area(60,82,760,20),
            center)),
            send(@obj_heading, font, @font),
            send(@main, display, @obj_heading),

            % create and display objectives list
            goal(GOAL),
            new(@obj_text, string),
            showlist(@obj_text, GOAL, state),
        send(@obj_text, append, '.'),
        new(@objective, text_block(@obj_text, area(60,100,760,80), center)),
        send(@main, display, @objective),

        % create and display current states heading
        new(@facts_heading, text_block('The following facts are now true:',
        area(60,192,760,20), center)),
        send(@facts_heading, font, @font),
        send(@main, display, @facts_heading),

        % create and display operator box heading
        new(@op_heading, text_block('Select an action: ',
        area(60,710,760,20), center)),
        send(@op_heading, font, @font),
        send(@main, display, @op_heading),

        % create operator box
        new(@operator_box, box(288,740,300,40)),
        send(@operator_box, pen, 2),
        send(@main, display, @operator_box),

        % create display bitmap with border
        clear(@display),
        clear(@b),
        new(@display, bitmap(926, 380)),
        new(@b, box(10,298,910,384)),
        send(@main, draw_in, @b),
        send(@display, position, point(12, 300)),
        send(@main, display, @display),
        % display initial state and operators, run the tutor.
        start_state(STATE),
        display_facts(STATE),
        retrieve_picture(STATE),
        display_operator_list.


% -------------------------------------------------------------------------------
% Display intro, intro text is created by the instructor.
% -------------------------------------------------------------------------------
display_intro :-

        % create and display intro heading
        new(@intro_heading, text_block('Introduction', area(70,12,760,15), center)),
        send(@intro_heading, font, @font),
        send(@main, display, @intro_heading),

        % display intro text
        intro(Text),
        new(@intro, text_block(Text, area(70,15,760,60), center)),
        send(@main, display, @intro).
```

```
% ------------------------------------------------------------------------
% Display instructor's responses.
% ------------------------------------------------------------------------
% display response text
display_instructor(Text_Block) :-
        clear(@instructor),
        new(@instructor, text_block(Text_Block, area(70,780,760,60), center)),
        send(@main, display, @instructor).

write_instructor(Text) :-
        clear(@i),
        new(@i, string),
        send(@i, append, Text).


% ------------------------------------------------------------------------
% Menu for operator selection
% ------------------------------------------------------------------------
 % browser for operator selection
display_operator_list :-
        find_operators(OL),
        randperm(OL, POL),
        down_list(POL),
        send(@oplist, append, 'Exit'),
        send(@oplist, selected, cascade(@oplist, operator, 0)),
        send(@oplist, clicked, 0).

down_list([]).

down_list(POL) :-
        % process the first member of the list
        first(X, POL1, POL),

        % process the operator, example form: 'go fire'.
        clear(@op),
        new(@op, string),
        showlist(@op, [X], op),
        send_list(@oplist, append, @op),

        % build the translation table
        % example form: translate('go fire', go(fire)).
        get(@op, text, Text),
        F =.. [translate, Text, X],
        uniqueassert(F),
        down_list(POL1).
```

```
% ------------------------------------------------------------------------------------------
% This is the initialization of the means-ends tutor. It first runs quick error checks
% on the teacher's definitions, then assert some useful global variables, checks to
% verify  that the problem given is solvable.
% ------------------------------------------------------------------------------------------

initialize_tutor :-
            start_state(STATE),
            goal(GOAL),
            not(check_obvious_errors),
            issue_warnings,
            uniqueassert(top_goal(GOAL)),
            find_operators(XL),
            uniqueassert(op_list(XL)),
            write('Wait a moment while I analyze the problem thoroughly.'), nl,
            once_means_ends(STATE,GOAL,OPLIST2,GOALSTATE2),
            uniqueassert(top_solution(OPLIST2)),
            abolish(mainline_states/4),
            retract(session_num(NN)),
            NNp1 is NN+1,
            asserta(session_num(NNp1)),
            abolish(tutor_says_done/1),
            abolish(error_num/1),
            asserta(error_num(1)),
            uniqueassert(current_state(STATE)), !.

initialize_tutor(STATE,GOAL) :-
            write('The problem you gave me seems impossible.'), nl, !.

% ------------------------------------------------------------------------------------------
% This is a temporary top-level program to call tutor_selected repeatedly.
% ------------------------------------------------------------------------------------------
run :-
            tutor_says_done(S), !.

run :-
            send(@oplist, selected, cascade(@oplist, operator, 0)),
            send(@oplist, clicked, 0).

% operator selection
operator(@oplist, 'Exit') :-
            clear_all.

operator(@oplist, Text) :-

            % display selection
            clear(@op),
            new(@op, text_block(Text, area(300,740,302,36), center)),
            send(@main, display, @op),
            translate(Text, Operator),
            tutor(Operator).
```

```
% ---------------------------------------------------------------------------
% display the sates
% ---------------------------------------------------------------------------
display_facts(STATE) :-

            % display facts
            clear(@facts_text),
            clear(@facts),
            new(@facts_text, string),
            showlist(@facts_text, STATE, state),
            send(@facts_text, append, '.'),
            new(@facts, text_block(@facts_text, area(70,210,760,60), center)),
            send(@main, display, @facts).


% ---------------------------------------------------------------------------
% Subroutines for intialization of the tutor
% ---------------------------------------------------------------------------
% creating a list of operators
find_operators(XL) :-
            nice_bagof(X, P^precondition(X,P), XL1),
            nice_bagof(X, C^P^precondition(X,C,P), XL2),
            append(XL1, XL2, XL).


% ---------------------------------------------------------------------------
% Problem-definition errors: errors by the instructor building a particular
% means_ends tutor.
% ---------------------------------------------------------------------------
check_obvious_errors :-
            not(studentflag),
            setof([M,A],obvious_error(M,A),MAL), !,
            writepairlist(MAL)

obvious_error('a fact predicate name is misspelled: ',W2) :-
            member(W,[recommended,precondi-
tion,deletepostcondition,addpostcondition,
            randsubst,nopref,intro]),
            get_misspelling(W,W2), (P=..[W2,X,Y]; P=..[W2,X,Y,Z]; P=..[W2,X,-
Y,Z,R]),
            call(P),
            not(same(W2,xnopref)).

obvious_error('precondition fact missing for action ',O) :-
            recommended(D,O),
            not(get_precondition(O,S,L)).

obvious_error('deletepostcondition fact missing for action ',O) :-
            recommended(D,O),
            not(get_deletepostcondition(O,S,L)).
```

```
obvious_error('addpostcondition fact missing for action ',O) :-
            recommended(D,O),
            not(get_addpostcondition(O,S,L)).

obvious_error('recommended fact missing for action ',O) :-
            get_precondition(O,S,L),
            not(recommended(D,O)).

obvious_error('recommended fact missing for action ',O) :-
            get_deletepostcondition(O,S,L),
            not(recommended(D,O)).

obvious_error('recommended fact missing for action ',O) :-
            get_addpostcondition(O,S,L),
            not(recommended(D,O)).

issue_warnings :-
            not(studentflag),
            setof([M,A],possible_error(M,A),MAL), !,
            write('Warnings:'), nl,
            writepairlist(MAL), nl.

issue_warnings.

possible_error('This fact is not creatable: ',F) :-
            get_precondition(O,S,PL),
            member(F,PL), (atom(F); not(F=..[not|_])),
            uncreatable(F).

possible_error('This fact is not removable: ',F) :-
            get_precondition(O,S,PL),
            member(not(F),PL),
            unremovable(F).


% ------------------------------------------------------------------------
% Misspelling confirmation given two bound arguments
% ------------------------------------------------------------------------
fixspell(W1,W2) :-
            atom(W1),
            atom(W2), !,
            name(W1,AW1),
            fixspell2(AW1,AW2),
            name(W2,AW2).

fixspell(W1,W2) :-
            W1=..[P1|L],
            W2=..[P2|L],
            not(P1 = P2), !,
            fixspell(P1,P2).
fixspell(W1,W2) :-
          W1=..[P,Q1|L],
          W2=..[P,Q2|L],
          not(Q1=Q2), !,
          fixspell(Q1,Q2).

fixspell(W1,W2) :-
          W1=..[P,Q,R1|L],
          W2=..[P,Q,R2|L],
          not(R1 = R2), !,

fixspell(R1,R2).

fixspell2(AW,AW2) :-
          deleteone(X,AW,AW2).

fixspell2(AW,AW2) :-
          deleteone(X,AW2,AW).

fixspell2(AW,AW2) :-
          transpose(AW,AW2).


% ------------------------------------------------------------------------
% Computing possible misspellings in the teacher's program
% ------------------------------------------------------------------------
get_misspelling(W1,W2) :-
          name(W1,AW1),
          (deleteone(X,AW1,NAW1);
          deleteone(X,NAW1,AW1);
          transpose(AW1,NAW1)),
          lettercode(X),
          name(W2,NAW1),
          not(same(W1,W2)).

lettercode(X) :-
          member(X,[97,98,99,100,101,102,103,104,105,106,107,108,
          109,110,111,112,113,114,115,116,117,118,119,120,121,122]).

transpose([X,Y|L],[Y,X|L]).

transpose([X|L],[X|M]) :-
          transpose(L,M).


writepairlist([]).

writepairlist([[X,Y]|L]) :-
          write(X), write(Y), nl,
          writepairlist(L).
```

```
% ---------------------------------------------------------------------------
% Handling of randomness
%
% After the postconditions are applied, random substitution (randsubst) definitions are
% applied to the state. These can add facts, delete facts, or change facts. The first
% argument to 'randsubst' is the operator involved, and the second argument is a list of
% quadruples. The first argument of each quadruple is the fact being matched, the second
% argument is the fact it should be replaced with, the third is the probability of this
% change, and the optional fourth argument is a message printed at the time this change
% is done. Either of the first two arguments can be 'none' to allow additions and
% deletions.
% ---------------------------------------------------------------------------

do_randsubst(O,S,NS) :-
        randsubst(O,RL), !,
        do_randsubst2(RL,S,NS).

do_randsubst(O,S,S).

do_randsubst2([],S,S).

do_randsubst2([[F,NF,P]|L],S,NS) :-
        random(X),
        X<P,
        changestate(F,NF,S,S2), !,
        do_randsubst2(L,S2,NS).

do_randsubst2([[F,NF,P,M]|L],S,NS) :-
        random(X),
        X<P,
        changestate(F,NF,S,S2), !,
        write_instructor(M),
        do_randsubst2(L,S2,NS),
        display_instructor(@i).
do_randsubst2([C|L],S,NS) :-
        do_randsubst2(L,S,NS).

changestate(none,NF,S,[NF|S]) :- !,
        not(member(NF,S)),
        write_instructor('Random change made: fact '),
        showfact(@i,NF,state),
        send(@i, append, ' added.'),
        display_instructor(@i), !.

changestate(F,none,S,S2) :- !,
        member(F,S),
        delete(F,S,S2),
        write_instructor('Random change made: fact '),
        showfact(@i, NF,state),
        send(@i, append, ' removed.'),
        display_instructor(@i), !.
```

```
changestate(F,NF,S,[NF|S3]) :- !,
            member(F,S),
            delete(F,S,S3),
            writeinstructor(@i, 'Random change made: fact '),
            showfact(@i, NF,state),
            send(@i, append, ' added, and fact '),
            showfact(@i,F,state),
            send(@i, append, ' removed.'),
            display_instructor(@i), !.

randperm([],[]) :- !.

randperm(L,[I|PL]) :-
            randitem(L,I),
            delete(I,L,L2),
            randperm(L2,PL).

randitem(L,I) :-
            length(L,N),
            Np1 is N+1,
            random(1,Np1,K),
            item(K,L,I).


% ------------------------------------------------------------------------
% Tutoring rules
% This manages the tutoring and simulation at each student selection.
% ------------------------------------------------------------------------

tutor(Op) :-
            current_state(S),
            top_goal(G),
            once_means_ends(S,G,[TutorOp|TOL],FS), !,
            tutor2(Op,S,G,TutorOp), !.

tutor(Op) :-
            write_instructor('I cannot solve the problem anymore.'),
            display_instructor(@i).


tutor2(Op,S,G,TutorOp) :-
            writedebug8(TutorOp),
            get_difference(G,S,D),
            handle_student_op(Op,TutorOp,S,D,FinalOp), !,
            talky_apply_op(FinalOp,S,S2),
            do_randsubst(FinalOp,S2,NewS),
            uniqueassert(current_state(NewS)),
            check_mainline_return(NewS),
            check_if_done(NewS,G),
            display_facts(NewS),
        % retrieve picture here
        retrieve_picture(NewS), !.


tutor2(Op,S,G,TutorOp).

check_if_done(S,G) :-
        get_difference(G,S,[]),
        write_instructor('Congratulations! You are done.'),
        asserta(tutor_says_done(S)),
        display_instructor(@i), !.

check_if_done(S,G) :- !.


% -------------------------------------------------------------------------------
% retrieve pictures according to state
% -------------------------------------------------------------------------------
retrieve_picture(State) :-
        send(@display, clear),

        % make a list of facts in the filename form (eg. fire_is_raging).
        convert(State, Filenames),
        filelist(F),
        retrieve(F).

convert([], _).

convert([S|S1], Filenames) :-

        % make the filename, example form: 'fire_is_location'.
        clear(@fn),
        new(@fn, string),
        showlist(@fn, [S], statefile),
        get(@fn, text, Text),
        % put it in the filename list
        append(Filenames, [Text], Filenames1),
        uniqueassert(filelist(Filenames1)),

        % process next member of the list
        convert(S1, Filenames1).


retrieve([]) :- !.

% retrieve the new state's pictures
retrieve([Filename|Rest]) :-
        display_picture(Filename),
        retrieve(Rest).

% retrieve only if the file is there, otherwise just continue.
```

```
display_picture(Filename) :-

            % retrieve if bitmap file exists
            exist(Filename) ->
            (area(Filename, X, Y, W, H),
            new(Bitmap, bitmap(W, H)),
            send(Bitmap, load, Filename),
            send(Bitmap, position, point(X, Y)),
            send(@display, draw_in, Bitmap))|
            true.


% -----------------------------------------------------------------------------------
% This implements the tutoring strategies for different kinds of student errors. Some
% are straightforward like spelling errors and precondition violations, and cause failure.
% Others require complex analysis with calls to 'means_ends' on hypothetical states
% created by the student. Some rules notice that something is wrong, but let the student
% proceed after a warning because it is probably better teaching strategy in this case to
% let the student find out for themselves the negative consequences of their action
% selection. One rule does not tutor immediately, but sets up a flag in the database that
% the student seems to be pursuing a digression, and tutors when the student returns from
% that digression via the 'check_mainline_return' line in 'tutor2'.
% -----------------------------------------------------------------------------------

handle_student_op(O2,O,S,D,NO) :-
            op_list(OL),
            not(member(O2,OL)),
            write_instructor( 'Not a valid action.'),
            display_instructor(@i), !, fail.

% Record your choice and student's choice before checking any more rules.
handle_student_op(O2,O,S,D,NO) :-
            not(same(O2,O)),
            not(xnopref(O2,O)),
            session_num(N1),
            error_num(N2),
            top_goal(G),
            asserta(student_error(N1,N2,O2,O,S,G)),
            N2p1 is N2+1,
            retract(error_num(N2)),
            asserta(error_num(N2p1)), fail.

% Will student operator not change the state?
handle_student_op(O2,O,S,D,NO) :-
            useless_op(O2,S),
            get_addpostcondition(O2,S,[PA]),
            get_deletepostcondition(O2,S,[PD]), !,
            write_instructor( 'It is already true that '),
            showfact(@i, PA,state),
            send(@i, append, '.'),
            display_instructor(@i).
```

```
handle_student_op(O2,O,S,D,O2) :-
        xuseless_op(O2,S), !,
        write_instructor(
        'That will not directly affect anything, but let us try it anyway.'),
        display_instructor(@i).

% Will student action lead to unsolvability of the problem?
handle_student_op(O2,O,S,D,NO) :-
        disaster_op(O2,S), !,
        write_instructor( 'You cannot ever succeed if you do that!'),
        display_instructor(@i), !, fail.

% Is student operator same as tutor operator? Then return.
handle_student_op(O,O,S,D,O) :- !,
        write_instructor( 'OK!'),
        display_instructor(@i).

% Or is student operator ranked as appropriate as tutor operator?
handle_student_op(O2,O,S,D,O2) :-
        xnopref(O2,O), !,
        write_instructor( 'OK!'),
        display_instructor(@i).


% Has the student ignored the tutor's operator 5 times before? Then gripe.
handle_student_op(O2,O,S,D,NO) :-
        session_num(N1),
        bagof(N2,O3^S3^G^student_error(N1,N2,O3,O,S3,G),N2L),
        length(N2L,M),
        T is M mod 5, T=0,
        write_instructor('Say, why not do the '),
        showfact(@i,O,op),
        send(@i, append, ' action?'),
        display_instructor(@i), fail.

% Could student have confused this operator with another? Then warn.
handle_student_op(O2,O,S,D,NO) :-
        confusable(O2,O,S),
        write_instructor( 'Warning: maybe you confused that with the '),
        showfact(@i,O,op),
        send(@i, append, ' action?'),
        display_instructor(@i), fail.


handle_student_op(O2,O,S,D,NO) :-
        xnopref(O,O3),
        confusable(O2,O3,S),
        desirable_op(D,O3),
        write_instructor( 'Warning: maybe you confused that with the '),
        showfact(@i,O3,op),
        send(@i, append, ' action?'),
                display_instructor(@i), fail.

% Could student have misread a fact in the state description? Then warn.
handle_student_op(O2,O,S,D,NO) :-
                get_precondition(O2,S,PO2),
                get_difference(PO2,S,[P]),
                member(P2,S), confusable(P,P2,S),
                write_instructor( 'Warning: maybe you confused "'),
                showfact(@i, P,state),
                send(@i, append, '" with "'),
                showfact(@i,P2,state),
                send(@i, append, '"?'),
                display_instructor(@i), fail.

% Does student's operator violate preconditions? Then ask for new one.
handle_student_op(O2,O,S,D,NO) :-
                get_precondition(O2,S,PO2),
                get_difference(PO2,S,D2),
                not(D2=[]), !,
                write_instructor( 'That action requires that '),
                showlist(@i ,D2,precond),
                send(@i, append, '.'),
                display_instructor(@i), !, fail.

% If student seems to be digressing, make a note for future reference.
handle_student_op(O2,O,S,D,O2) :-
                top_goal(G),
                apply_op(O,S,S3),
                apply_op(O2,S,S2),
                compare_solutions(S3,G,OL3,GS3,S2,G,OL2,GS2),
                subsequence([O|OL3],OL2), !,
                apply_ops([O|OL3],S,SL,GS4),
                elimdups(SL,ESL),
                asserta(mainline_states(ESL,O2,S,O)),
                write_instructor( 'Your action does not seem immediately helpful,
                but I will try it.'),
                display_instructor(@i).

% Grumble if student's operator will never help solve the problem.
handle_student_op(O2,O,S,D,O2) :-
                top_goal(G),
                once_means_ends(S,G,OL,FS),
                not(member(O2,OL)), !,
                write_instructor( 'OK, but I am not sure you need to do that action.'),
                display_instructor(@i).

% Grumble if student's operator is not the highest-recommended.
```

```
handle_student_op(O2,O,S,D,O2) :-
            top_goal(G), get_difference(G,S,D2),
            once_means_ends(S,D2,_,_),
            desirable_op(D2,O3),
            get_precondition(O3,S,PL),
            least_common_op(S,G,O,O2,PL,GROOT), !,
            write_instructor( 'OK, but that action will not help achieve these
            desirable things: '),
            get_difference(GROOT,S,D5),
            delete_uncreatable(D5,D6),
            randperm(D6,D7),
            showlist(@i,D7,precond),
            send(@i, append, '.'),
            display_instructor(@i).
% Else grumble because you can't understand what student is doing.
handle_student_op(O2,O,S,D,O2) :-
            write_instructor( 'Your action is not what I would choose,
            but let us try it.'),
            display_instructor(@i), !


% ------------------------------------------------------------------------------
% Intermediate predicates used by the tutor
% ------------------------------------------------------------------------------
xnopref(O1,O2) :-
            nopref(O1,O2).
xnopref(O1,O2) :-
            nopref(O2,O1).

useless_op(O,S) :-
            apply_op(O,S,S), !
xuseless_op(O,S) :-
            apply_op(O,S,S),
            not(randsubst(O,_)), !.
disaster_op(O2,S) :-
            cached_disaster_op(O2,S), !.
disaster_op(O2,S) :-
            apply_op(O2,S,S2),
            top_goal(G),
            not(once_means_ends(S2,G,_,_)),
            asserta(cached_disaster_op(O2,S)), !.
desirable_op(D,O) :-
            recommended(D2,O),
            subset(D2,D).
```

```
% ------------------------------------------------------------------------
% This is used when the student has picked an operator which does help solve the problem
% but is not the highest-priority operator (i.e., he has a bug in his internal 'recommend-
% ed' definitions.). It tries to find a goal that the student could be working on that
% explains his choice of the wrong operator, and tries to tailor its explanation to what
% should be done first to achieve that goal.
% ------------------------------------------------------------------------
least_common_op(S,G,O,O2,G2,G) :-
        once_means_ends(S,G2,OL,NS),
        least_common_op2(O,O2,OL).
least_common_op(S,G,O,O2,G2,DROOT) :-
        get_difference(G2,S,D),
        once_means_ends(S,D,_,_),
        desirable_op(D,O3),
        get_precondition(O3,S,G3),
        least_common_op(S,G2,O,O2,G3,DROOT), !.

least_common_op2(O,O2,OL) :-
        not(member(O,OL)), !.
least_common_op2(O,O2,OL) :-
        not(member(O2,OL)), !.

compare_solutions(S3,G,OL3,GS3,S2,G,OL2,GS2) :-
        once_means_ends(S3,G,OL3,GS3),
        once_means_ends(S2,G,OL2,GS2), !.

% ------------------------------------------------------------------------
% Since the tutor repeatedly reexamines slightly different paths to the goal, a lot of
% redundancy can be avoided by having the tutor store every solution it has found (by
% 'means_ends') to a problem. And fact order shouldn't matter in caching states.
% ------------------------------------------------------------------------
cache_states(S,G,[],GS) :- !.
cache_states(S,G,OL,GS) :-
        cached(S,G,OL,GS), !.
cache_states(S,G,OL,GS) :-
        cached(S2,G2,OL2,GS2),
        check_permutation(S,S2),
        check_permutation(G,G2), !.
cache_states(S,G,[O|OL],GS) :-
        asserta(cached(S,G,[O|OL],GS)),
        apply_op(O,S,NS),
        cache_states(NS,G,OL,GS), !.

% ------------------------------------------------------------------------
% This takes a list of operators and tells you what the resulting state is after applying
% them to some starting state.
% ------------------------------------------------------------------------
apply_ops([],S,[S],S) :- !.
apply_ops([O|OL],S,[S|SL],NS) :-
        apply_op(O,S,S2),
        apply_ops(OL,S2,SL,NS).
```

```
apply_op(O,S,NS) :-
            get_precondition(O,S,PCL),
            get_difference(PCL,S,[]),
            get_deletepostcondition(O,S,DP),
            deleteitems(DP,S,S2),
            get_addpostcondition(O,S,AP),
            union(AP,S2,NS), !.

talky_apply_op(O,S,NS) :-
            get_precondition(O,S,PCL),
            get_difference(PCL,S,[]),
            get_deletepostcondition(O,S,DP),
            deleteitems(DP,S,S2),
            get_addpostcondition(O,S,AP),
            union(AP,S2,NS),
            print_optional_message_d(O,S),
            print_optional_message_a(O,S), !.


% ------------------------------------------------------------------------------------------
% This checks for when the student returns from a digression, so as to tutor him at that
% point.
% ------------------------------------------------------------------------------------------
check_mainline_return(S) :-
            mainline_states(SL,O,OS,BO),
            check_mainline_return2(S,SL,O,OS,BO).
check_mainline_return(S).

check_mainline_return2(S,[S2|SL],O,OS,BO) :-
            permutemember(S,[S2]), !,
            write_instructor('You are returning to a previous state.'),
            display_instructor(@i).

check_mainline_return2(S,SL,O,OS,BO) :-
            permutemember(S,SL), !,
            write_instructor( 'Do you see now that your choice of the '),
            showfact(@i,O,op),
            send(@i, append, ' action in the state with the facts ['),
            showlist(@i,OS,state),
            send(@i, append, '] was not the best choice; the '),
            showfact(@i,BO,op),
            append(@i, ' action would have been better.'),
            retract(mainline_states(SL,O,OS,BO)),
            display_instructor(@i).

confusable(O,O,S) :- !, fail.

% Two actions are confusable if they do the opposite things
confusable(O1,O2,S) :-
            get_deletepostcondition(O1,S,DL1),
            get_deletepostcondition(O2,S,DL2),
          get_addpostcondition(O1,S,DL2),
          get_addpostcondition(O2,S,DL1), !.

% Two actions or literals are confusable if their first word is identical
confusable(O1,O2,S) :-
          O1=..[P|R1], O2=..[P|R2], !.

% Or if they are words whose first two letters are identical,
% or where you can delete first one or two letters to get the other word
confusable(O1,O2,S) :-
          atom(O1), atom(O2),
          name(O1,[C1,C2|NO1]), name(O2,[C3,C4|NO2]),
          (same(NO1,NO2);
          same(NO1,[C3,C4|NO2]);
          same(NO2,[C1,C2|NO1]);
          same([C2|NO1],[C3,C4|NO2]);
          same([C4|NO2],[C1,C2|NO1]) ), !.


% ------------------------------------------------------------------------------
% The original means-ends program (used for 'what if' reasoning)
%
% Note that this works little differently from 'means_ends_tutor' in that it checks for
% infinite loops for several situations that the earlier definition does not. A goal-state
% stack is kept to check new goals and states against. Random substitution via 'randsubst'
% is ignored, so in fact the solution paths found by this program may be quite different
% from those typically encountered in the tutor, and in some pathological cases with
% probabilities of 1 or 0 the results of this program may be in fact impossible, though
% that is unlikely.
% ------------------------------------------------------------------------------

once_means_ends(STATE,GOAL,OPLIST,GOALSTATE) :-
          means_ends(STATE,GOAL,OPLIST,GOALSTATE),
          cache_states(STATE,GOAL,OPLIST,GOALSTATE), !.

means_ends(STATE,GOAL,OPLIST,GOALSTATE) :-
          means_ends2(STATE,GOAL,OPLIST,GOALSTATE,[]),
          writedebug7.

means_ends2(STATE,GOAL,OPLIST,GOALSTATE,STACK) :-
          cached(STATE2,GOAL2,OPLIST,GOALSTATE),
          check_permutation(GOAL,GOAL2),
          check_permutation(STATE,STATE2), !,
          writedebug6(STACK), !.

means_ends2(STATE,GOAL,OPLIST,GOALSTATE,STACK) :-
          member([STATE,GOAL],STACK), !,
          writedebug4(STATE,GOAL,STACK), fail.

means_ends2(STATE,GOAL,[],STATE,STACK) :-
          get_difference(GOAL,STATE,[]), !.
```

```
means_ends2(STATE,GOAL,OPLIST,GOALSTATE,STACK) :-
          get_difference(GOAL,STATE,D),
          desirable_op(D,OPERATOR),
          get_precondition(OPERATOR,STATE,PRELIST),
          all_achievable(STATE,PRELIST),
          writedebug1(D,OPERATOR,STACK),
          means_ends2(STATE,PRELIST,PREO-
PLIST,PRESTATE,[[STATE,GOAL]|STACK]),
          writedebug2(PRESTATE,D,OPERATOR,STACK),
          get_deletepostcondition(OPERATOR,PRESTATE,DELETEPOSTLIST),
          deleteitems(DELETEPOSTLIST,PRESTATE,PRESTATE2),
          get_addpostcondition(OPERATOR,PRESTATE,ADDPOSTLIST),
          union(ADDPOSTLIST,PRESTATE2,POSTLIST),
          means_ends2(POSTLIST,GOAL,-
POSTOPLIST,GOALSTATE,[[STATE,GOAL]|STACK]),
          writedebug3(GOALSTATE,OPERATOR,STACK),
          append(PREOPLIST,[OPERATOR|POSTOPLIST],OPLIST).

means_ends2(STATE,GOAL,OPLIST,GOALSTATE,STACK) :-
          writedebug5(STATE,GOAL,STACK), !, fail.



% -----------------------------------------------------------------------
% Debugging tools
%
% These are enabled when the user asserts the no-argument predicate 'debugflag'.
% -----------------------------------------------------------------------
writedebug1(D,O,STACK) :-
          not(debugflag), !.

writedebug1(D,O,STACK) :-
          length(STACK,NM1),
          N is NM1+1,
          write('>>Action '),
          write(O),
          write(' suggested at level '),
          write(N),
          write('to achieve difference of ['),
          write(D,state),
          write(']'),
          !, nl.

writedebug2(S,D,O,STACK) :-
          not(debugflag), !.

writedebug2(S,D,O,STACK) :-
          length(STACK,NM1), N is NM1+1,
          write( '>>Action '),
          write(O),
          write(' applied at level '),
          write(N),
          write('to reduce difference of ['),
          showlist(D,state),
          write(']'),
          write('in state in which '),
          writelist(S,state),
          !, nl.

writedebug3(S,O,STACK) :-
          not(debugflag), !.

writedebug3(S,O,STACK) :-
          length(STACK,NM1),
          N is NM1+1,
          write( '>>Level '),
          write(N),
          write(' terminated at state in which '),
          writelist(S,state),
          !, nl.

writedebug4(S,G,STACK) :- not(debugflag), !.

writedebug4(S,G,STACK) :-
          write( '>>>>Reasoning avoided an infinite loop at level '),
          length(STACK,NM1),
          N is NM1+1,
          write(N),
          write(' where problem was identical to that at level '),
          index([S,G],STACK,I),
          write(I),
          !, nl.

writedebug5(STATE,GOAL,STACK) :-
          not(debugflag), !.

writedebug5(STATE,GOAL,STACK) :-
          write( '>>>>Unsolvable problem at level '),
          length(STACK,NM1),
          N is NM1+1,
          write(N), nl,
          write(' for state '),
          writelist(STATE,state),
          write(' and goal '),
          writelist(GOAL,state),
          nl.

writedebug6(STACK) :-
          not(debugflag), !.

writedebug6(STACK) :-
          write( '>>>>Previously computed solution used at level '),
        length(STACK,NM1), N is NM1+1,
        write(N),
        !, nl.

writedebug7 :-
        not(debugflag), !.

writedebug7 :- nl, !.

writedebug8(OP) :-
        not(debugflag), !.

writedebug8(OP) :-
        write( 'The tutor prefers action '),
        writefact(OP,op),
        !, nl.

flag_errors :-
        uniqueassert(debugflag).

unflag :-
        retract(debugflag).

% ---------------------------------------------------------------------------
% Miscellaneous utility functions
% ---------------------------------------------------------------------------

all_achievable(S,G) :-
        get_difference(G,S,D),
        not(unachievable_member(D)).

delete_uncreatable([],[]).

delete_uncreatable([X|L],M) :-
        uncreatable(X), !,
        delete_uncreatable(L,M).

delete_uncreatable([X|L],[X|M]) :-
        delete_uncreatable(L,M).

unachievable_member(D) :-
        member(F,D),
        (atom(F); not(F=..[not|_])),
        uncreatable(F), !.

unachievable_member(D) :-
        member(not(F),D),
        unremovable(F), !.

uncreatable(F) :-
        not(in_addpostcondition(F)), !.
```

```
unremovable(F) :-
            not(in_deletepostcondition(F)), !.

in_deletepostcondition(F) :- !,
            any_deletepostcondition(O,L),
            member(F,L), !.

in_addpostcondition(F) :-
            any_addpostcondition(O,L),
            member(F,L), !.

added_by_randsubst(F) :-
            randsubst(O,RSL),
            member([_,F|_],RSL), !.

deleted_by_randsubst(F) :-
            randsubst(O,RSL),
            member([F|_],RSL), !.

any_addpostcondition(O,L) :-
            addpostcondition(O,C,L,M).

any_addpostcondition(O,L) :-
            addpostcondition(O,C,L)

any_addpostcondition(O,L) :-
            addpostcondition(O,L).
any_deletepostcondition(O,L) :-
            deletepostcondition(O,C,L,M).

any_deletepostcondition(O,L) :-
            deletepostcondition(O,C,L).

any_deletepostcondition(O,L) :-
            deletepostcondition(O,L).


get_deletepostcondition(O,S,L) :-
            deletepostcondition(O,C,L,M),
            factsubset(C,S), !.

get_deletepostcondition(O,S,L) :-
            deletepostcondition(O,C,L),
            factsubset(C,S), !.

get_deletepostcondition(O,S,L) :-
            deletepostcondition(O,L).


get_addpostcondition(O,S,L) :-
```

```
            addpostcondition(O,C,L,M),
            factsubset(C,S), !.

get_addpostcondition(O,S,L) :-
            addpostcondition(O,C,L),
            factsubset(C,S), !.

get_addpostcondition(O,S,L) :-
            addpostcondition(O,L).


get_precondition(O,S,L) :-
            precondition(O,C,L,M),
            factsubset(C,S), !.

get_precondition(O,S,L) :-
            precondition(O,C,L),
            factsubset(C,S), !.

get_precondition(O,S,L) :-
            precondition(O,L).


print_optional_message_d(O,S) :-
            deletepostcondition(O,C,L,M),
            factsubset(C,S),
            write_instructor(M), !,
            display_instructor(@i).

print_optional_message_d(O,S) :- !.

print_optional_message_a(O,S) :-
            addpostcondition(O,C,L,M),
            factsubset(C,S),
            write_instructor(M), !,
            display_instructor(@i).

print_optional_message_a(O,S) :- !.


% ---------------------------------------------------------------------------------
% Freeing objects
% ---------------------------------------------------------------------------------

clear_all :-
            clear(@main),
            clear(@display),
            clear(@bit),
            clear(@oplist),
            clear(@operator_box),
            clear(@objective),
        clear(@obj_heading),
        clear(@obj_text),
        clear(@facts_heading),
        clear(@op_heading),
        clear(@intro),
        clear(@intro_heading),
        clear(@font),
        clear(@font2).

% Eof MeGraph21
```

APPENDIX E

SOURCE CODE OF COMMON

```
%=========================================================================
% Program        : common
% Purpose        :This program is a collection of general procedures used in common by
%                 DrawGraph and MeGraph21.
% Author         : Francius Suwono
% Date           : April 6, 1992
%-------------------------------------------------------------------------

%-------------------------------------------------------------------------
% dialog and browser for geting filename
%-------------------------------------------------------------------------
select_filename(Dialog) :-

        % make the dialog
        new_dialog(Dialog, 'Select Filename', select_name),

        % make the browser
        clear(@browser),
        new(@browser, browser('Filename selection')),
        send(@browser, size, size(250,150)),
        send(@browser, selected, cascade(@browser, filename, 0)),
        send(@browser, clicked, 0),

        % fill the browser with filenames from facts
        find_all_facts(FL),
        fill_browser(@browser, FL),

        % arrange the browser position, and open it
        send(@browser, above, Dialog),
        send(@browser, open),
        send(@browser, sort).

% callbacks
filename(Browser, Text) :-
        new(Filename, string),
        send(Filename, append, Text),
        send(@ti, selection, Filename).

% dialog declarations for getting filename
select_name(_, label('Click or type filename.'), below,[]).
select_name(@ti, text_item('Filename: ', '', 0), below, []).
select_name(_, button('OK', pressed), below, []).
select_name(_, button('Cancel', pressed), right, []).
```

```
%---------------------------------------------------------------------------
% derive filenames from facts, and put it in a browser window
%---------------------------------------------------------------------------
fill_browser(Browser, []).

fill_browser(Browser, [X|FL1]) :-

        % make the filename, example form: 'fire_is_location'.
        clear(@fn),
        new(@fn, string),
        showlist(@fn, [X], statefile),

        % put it in the browser list
        send_list(Browser, append, @fn),

        % process next member of the list
        fill_browser(Browser, FL1).


% creating a list of facts
find_all_facts(XL) :-
        nice_bagof(X,O^recommended(X,O),XL1),
        nice_bagof(X,O^addpostcondition(O,X), XL2),
        nice_bagof(X,O^precondition(O,X), XL3),
        start_state(XL4),
        append(XL1, XL2, X1),
        append(XL3, XL4, X2),
        append(X1, X2, X3),
        flatten(X3, X4),
        elimdups(X4, XL).


%---------------------------------------------------------------------------
% write all picture positions into posfile
%---------------------------------------------------------------------------
write_position :-
        tell(posfile),
        write_pos,
        told.

write_pos :-
        repeat,
        ((area(Filename, X, Y, W, H),
        write('area('),
        write(Filename),
        write(','),
        write(X),
        write(','),
        write(Y),
        write(','),
        write(W),
            write(','),
            write(H),
            write(')'),
            write('.'),
            nl,
            fail) | !).


% clear any object
clear(X) :-
            object(X) ->
            send(X,destroy) | true.

demolish(X) :-
            object(X) ->
            send(X, demolish) | true.

%------------------------------------------------------------------------------
% read all picture positions from posfile
%------------------------------------------------------------------------------
read_position :-
            see(posfile),
            process_position,
            seen.

process_position :-
            read(Term),
            process(Term).

process(end_of_file) :- !.

process(Term) :-
            uniqueassert(Term),
            process_position.

%------------------------------------------------------------------------------
% input file existence checking
%------------------------------------------------------------------------------
exist(Filename) :-
            see(Filename),
            nofileerrors,
            seen.


%------------------------------------------------------------------------------
%  Natural language processing
% These routines are not perfect, but they seem to work well most of the time.
% 'showlist' takes a third argument designating what kind of list it is, since
% these things are output differently depending on whether they are states,
% preconditioned lists, or operators. The output is a text object.
%------------------------------------------------------------------------------
```

```
showlist(Text, [],R) :- !.

showlist(Text, [X],R) :-
          !,
          showfact(Text, X,R).

showlist(Text, [X,Y],R) :-
          !,
          showfact(Text, X,R),
          send(Text, append, ' and '),
          showfact(Text, Y,R).

showlist(Text, L,R) :-
          showlist2(Text, L, R).

showlist2(Text, [X],R) :-
          !,
          showfact(Text, X,R).

showlist2(Text, [X|L],R) :-
          showfact(Text, X, R),
          send(Text, append, ', '),
          showlist2(Text, L,R).


%--------------------------------------------------------------------------------------------
% Prowindows version: output format for states.
%--------------------------------------------------------------------------------------------
showfact(Text, F,state) :-
          atom(F),
          send(Text, append, 'it is '),
          send(Text, append, F),
          !.

showfact(Text, not(F),state) :-
          atom(F),
          !,
          send(Text, append, 'it is not '),
          send(Text, append, F),
          !.


showfact(Text, not(F),state) :-
          F=..[P,X],
          atom(X),
          !,
          send(Text, append, X),
          is_form(X,IX),
          send(Text, append, ' '),
          send(Text, append, IX),
          send(Text, append, ' '),
            send(Text, append, 'not '),
            send(Text, append, P),
            !.


showfact(Text, not(F),state) :-
            F=..[P,X],
            !,
            send(Text, append, X),
            is_form(X,IX),
            send(Text, append, ' '),
            send(Text, append, IX),
            send(Text, append, ' not '),
            send(Text, append, P),
            !.

showfact(Text, not(F),state) :-
            F=..[status,X,Y],
            !,
            send(Text, append, 'the '),
            send(Text, append, X),
            send(Text, append, ' status '),
            is_form(Y,IY),
            send(Text, append, ' '),
            send(Text, append, IY),
            send(Text, append, ' not '),
            send(Text, append, Y),
            !.

showfact(Text, not(F),state) :-
            F=..[P,X,Y],
            !,
            send(Text, append, X),
            send(Text, append, ' '),
            send(Text, append, Y),
            is_form(Y,IY),
            send(Text, append, ' '),
            send(Text, append, IY),
            send(Text, append, ' not '),
            send(Text, append, P),
            !.

showfact(Text, F,state) :-
            F=..[P,X], atom(X),
            !,
            send(Text, append, X),
            is_form(X,IX),
            send(Text, append, ' '),
            send(Text, append, IX),
            send(Text, append, ' '),
            send(Text, append, P),
            !.

showfact(Text, F,state) :-
            F=..[P,X],
            !,
            showfact(Text, X,state),
            is_form(X,IX),
            send(Text, append, ' '),
            send(Text, append, IX),
            send(Text, append, ' '),
            send(Text, append, P),
            !.

showfact(Text, F,state) :-
            F=..[status,X,Y],
            !,
            send(Text, append, 'the '),
            send(Text, append, X),
            send(Text, append, ' status '),
            is_form(Y, IY),
            send(Text, append, ' '),
            send(Text, append, IY),
            send(Text, append, ' '),
            send(Text, append, Y),
            !.

showfact(Text, F,state) :-
            F=..[P,X,Y],
            !,
            send(Text, append, X),
            send(Text, append, ' '),
            send(Text, append, Y),
            is_form(Y,IY),
            send(Text, append, ' '),
            send(Text, append, IY),
            send(Text, append, ' '),
            send(Text, append, P),
            !.

%------------------------------------------------------------------------------
% Filename from facts
%------------------------------------------------------------------------------
showfact(Text, F,statefile) :-
            atom(F),
            send(Text, append, 'it_is_'),
            send(Text, append, F),
            !.

showfact(Text, not(F),statefile) :-
            atom(F),
            !,
            send(Text, append, 'it_is_not_'),
            send(Text, append, F),
            !.

showfact(Text, not(F),statefile) :-
            F=..[P,X],
            atom(X),
            !,
            send(Text, append, X),
            is_form(X,IX),
            send(Text, append, '_'),
            send(Text, append, IX),
            send(Text, append, '_not_'),
            send(Text, append, P),
            !.
showfact(Text, not(F),statefile) :-
            F=..[P,X],
            !,
            send(Text, append, X),
            is_form(X,IX),
            send(Text, append, '_'),
            send(Text, append, IX),
            send(Text, append, '_not_'),
            send(Text, append, P),
            !.


showfact(Text, not(F),statefile) :-
            F=..[status,X,Y],
            !,
            send(Text, append, 'the_'),
            send(Text, append, X),
            send(Text, append, '_status_'),
            is_form(Y,IY),
            send(Text, append, '_'),
            send(Text, append, IY),
            send(Text, append, '_not_'),
            !.


showfact(Text, not(F),statefile) :-
            F=..[P,X,Y],
            !,
            send(Text, append, X),
            send(Text, append, '_'),
            send(Text, append, Y),
            is_form(Y,IY),
            send(Text, append, '_'),
            send(Text, append, IY),
            send(Text, append, '_not_'),
            send(Text, append, P),
          !.


showfact(Text, F,statefile) :-
          F=..[P,X],
          atom(X),
          !,
          send(Text, append, X),
          is_form(X,IX),
          send(Text, append, '_'),
          send(Text, append, IX),
          send(Text, append, '_'),
          send(Text, append, P),
          !.


showfact(Text, F,statefile) :-
          F=..[P,X],
          !,
          showfact(Text, X,statefile),
          is_form(X,IX),
          send(Text, append, '_'),
          send(Text, append, IX),
          send(Text, append, '_'),
          send(Text, append, P),
          !.


showfact(Text, F,statefile) :-
          F=..[status,X,Y],
          !,
          send(Text, append, 'the_'),
          send(Text, append, X),
          send(Text, append, '_status_'),
          is_form(Y,IY),
          send(Text, append, '_'),
          send(Text, append, IY),
          send(Text, append, '_'),
          send(Text, append, Y),
          !.


showfact(Text, F,statefile) :-
          F=..[P,X,Y],
          !,
          send(Text, append, X),
          send(Text, append, '_'),
          send(Text, append, Y),
          is_form(Y,IY),
          send(Text, append, '_'),
          send(Text, append, IY),
            send(Text, append, '_'),
            send(Text, append, P),
            !.


%-----------------------------------------------------------------------
% Prowindows version: output for preconditions
%-----------------------------------------------------------------------
showfact(Text, F,precond) :-
            atom(F),
            send(Text, append, 'it must be '),
            send(Text, append, F),
            !.
showfact(Text, not(F),precond) :-
            atom(F),
            !,
            send(Text, append, 'it must not be '),
            send(Text, append, F),
            !.

showfact(Text, not(F),precond) :-
            F=..[P,X],
            !,
            showfact(Text, X,state),
            send(Text, append, ' must not be '),
            send(Text, append, P),
            !.

showfact(Text, not(F),precond) :-
            F=..[P,X,Y],
            !,
            send(Text, append, X),
            send(Text, append, ' '),
            send(Text, append, Y),
            send(Text, append, ' must not be '),
            send(Text, append, P),
            !.

showfact(Text, F,precond) :-
            F=..[P,X], atom(X),
            !,
            send(Text, append, X),
            send(Text, append, ' must be '),
            send(Text, append, P),
            !.

showfact(Text, F,precond) :-
            F=..[P,X],
            !,

            showfact(Text, X,state),
        send(Text, append, ' must be '),
        send(Text, append, P),
        !.

showfact(Text, F,precond) :-
        F=..[P,X,Y],
        send(Text, append, X),
        send(Text, append, ' '),
        send(Text, append, Y),
        send(Text, append, ' must be '),
        send(Text, append, P),
        !.


%----------------------------------------------------------------------------
% Prowindows version: output format for operators
%----------------------------------------------------------------------------
showfact(Text, F, op) :-
        F =.. [P, A],
        send(Text, append, P),
        send(Text, append, ' '),
        send(Text, append, A),
        !.

showfact(Text, F, op) :-
        F =.. [P, A, B],
        send(Text, append, P),
        send(Text, append, ' '),
        send(Text, append, A),
        send(Text, append, ' '),
        send(Text, append, B),
        !.

showfact(Text, F, op) :-
        send(Text, append, F),
        !.

showfact(Text, F, op) :-
        send(Text, append, F).


%----------------------------------------------------------------------------
% Prowindows version: output format for filename
%----------------------------------------------------------------------------
showfact(Text, F, fn) :-
        F =.. [P, A],
        send(Text, append, P),
        send(Text, append, '_'),
        send(Text, append, A),
        !.
```

```
showfact(Text, F, fn) :-
        F =.. [P, A, B],
        send(Text, append, P),
        send(Text, append, '_'),
        send(Text, append, A),
        send(Text, append, '_'),
        send(Text, append, B),
        !.

showfact(Text, F, fn) :-
        append(Text, F),
        !.

showfact(Text, F, fn) :-
        send(Text, append, F).


% A simple heuristic is used for plurals: the thing before the 'is'
% is plural if it ends in 's'.
is_form(X,'is') :-
        not(atom(X)),
        !.

is_form(X,'are') :-
        name(X,NX),
        last(NX,115),
        !.

is_form(X,'is').


%-------------------------------------------------------------------------------
% List utilities
%-------------------------------------------------------------------------------

% delete one item from a list
deleteone(X,[X|L],L).

deleteone(X,[Y|L],[Y|M]) :-
        deleteone(X,L,M).

% get the difference
get_difference([],S,[]).

get_difference([not(P)|G],S,G2) :-
        not(singlemember(P,S)), !,
        get_difference(G,S,G2).

get_difference([P|G],S,G2) :-
        singlemember(P,S), !,
        get_difference(G,S,G2).
get_difference([P|G],S,[P|G2]) :-
          get_difference(G,S,G2).


% test for a subset of a list
subset([],L).

subset([X|L],L2) :-
          singlemember(X,L2),
          subset(L,L2).

% test for facts subset
factsubset([],L).

factsubset([not(P)|L],L2) :-
          not(singlemember(P,L2)), !,
          factsubset(L,L2).

factsubset([not(P)|L],L2) :- !, fail.

factsubset([P|L],L2) :-
          singlemember(P,L2),
          factsubset(L,L2).


% test for member of a list
member(X,L) :-
          append(L1,[X|L2],L).


% test for a single member.
singlemember(X,[X|L]) :- !.

singlemember(X,[Y|L]) :-
          singlemember(X,L).
% unions two lists
union([],L,L).

union([X|L1],L2,L3) :-
          singlemember(X,L2), !,
          union(L1,L2,L3).

union([X|L1],L2,[X|L3]) :-
          union(L1,L2,L3).


% delete same items from a list.
deleteitems([],L,L).

deleteitems([X|L],L2,L3) :-
```

```
            delete(X,L2,L4),
            deleteitems(L,L4,L3).


% delete an item from a list
delete(X,[],[]).

delete(X,[X|L],M) :- !,
            delete(X,L,M).

delete(X,[Y|L],[Y|M]) :-
            delete(X,L,M).


% test for an item
item(K,[],I) :- !, fail.

item(K,[X|L],X) :-
            K=<1, !.

item(K,[X|L],Y) :-
            KM1 is K-1,
            item(KM1,L,Y).


% check permutation
check_permutation(L,M) :-
            subset(L,M),
            subset(M,L), !.


% check for subsequence
subsequence([],L) :- !.

subsequence([X|L],[X|M]) :- !,
            subsequence(L,M).

subsequence(L,[X|M]) :-
            subsequence(L,M).


% permute member of a list
permutemember(X,[X|L]) :- !.

permutemember(X,[Y|L]) :-
            subset(X,Y),
            subset(Y,X), !.

permutemember(X,[Y|L]) :-
            permutemember(X,L).
```

```
% last item of a list
last([X],X).

last([X|L],Y) :-
            last(L,Y).


% first item and tail of a list
first(X, Tail, L) :-
            append([X], Tail, L).


% eliminate duplicate items from a list
elimdups([],[]).

elimdups([X|L],M) :-
            singlemember(X,L), !,
            elimdups(L,M).

elimdups([X|L],[X|M]) :-
            elimdups(L,M).

% unique assert of fact.
uniqueassert(Q) :-
            retract(Q), !,
            asserta(Q).
uniqueassert(Q) :-
            asserta(Q).

% unique assert of picture position facts.
uniqueassert_area(P) :-
            P =.. [area, F, X, Y, W, H],
            retractall(area(F,_,_,_,_)),
            !,
            asserta(P).

uniqueassert_area(P) :-
            asserta(P).

index(X,[X|L],1) :- !.

index(X,[Y|L],N) :- index(X,L,Nm1), N is Nm1+1.

same(X,X).

nice_bagof(X,P,L) :- bagof(X,P,L), !.

nice_bagof(X,P,[]).

% to take out members of list
flatten([Head | Tail],FL) :-
```

```
            flatten(Head, FlatHead),
            flatten(Tail, FlatTail),
            append(FlatHead, FlatTail, FL).

flatten([], []).

flatten(X, [X]).

% min function
min(X, Y, Z) :-
            X > Y ->
            Z is Y | Z is X.

% not predicate
not(X)   :- \+ X.

% Eof common.
```

APPENDIX F

TEACHER'S DEFITIONS FOR THE FIRE FIGHTING TUTOR [ROWE 90]

```
%=====================================================================
% Program     : mefire
% Purpose     : This program is created  for tutoring fire fighting aboard ships
% Author      : Prof. Neil C. Rowe
% Date        : October 1989
% Source      : Naval Postgraduate School, Report NPS52-90-003, Means-Ends
%               Tutoring, Multi-Tutoring and Meta-Tutoring, by Neil C. Rowe,
%               p. 9, February 1990
% ---------------------------------------------------------------------

intro('You are the fire team leader on a U.S. Navy ship.  A fire has been reported.').

recommended([treated(casualty)], direct(medical, corpman)).
recommended([treated(casualty)], give(first, aid)).
recommended([not(present(casualty))], remove(casualty)).
recommended([not(unreplaced(casualty))], replace(casualty)).
recommended([equipped(team)], equip).
recommended([deenergized(fire, area)], deenergize).
recommended([set(boundaries)], set(boundaries)).
recommended([confronted(fire)], approach(fire)).
recommended([out(fire)], extinguish).
recommended([watched(reflashing)], set(reflash, watch)).
recommended([verified(out(fire))], verify(out)).
recommended([safe(gases)], test(gases)).
recommended([tested(oxygen, tester)], test(oxygen, tester)).
recommended([safe(oxygen)], test(oxygen)).
recommended([not(smokey)], desmoke).
recommended([estimated(water)], estimate(water)).
recommended([not(watery)], dewater).
recommended([not(equipped(team))], store(equipment)).
recommended([debriefed(team)], debrief).
recommended([not(watched(reflashing))], secure(reflash, watch)).
recommended([location(fire)], go(fire)).
recommended([location(repair, locker)], go(repair, locker)).
recommended([safe(X)], wait).

precondition(remove(casualty),  present(casualty), treated(casualty),
          not(dead(casualty))]).
precondition(direct(medical, corpman), [present(casualty),
          present(medical, corpman), not(dead(casualty))]).
precondition(give(first, aid), [present(casualty), not(present(medical, corpman)),
          not(dead(casualty))]).
precondition(replace(casualty), [unreplaced(casualty), not(present(casualty))]).
precondition(equip, [location(repair, locker), not(equipped(team))]).
precondition(deenergize, [location(fire)]).
precondition(set(boundaries), [location(fire), not(set(boundaries))]).
precondition(approach(fire), [location(fire), not(confronted(fire)), raging(fire),
          set(boundaries), equipped(team)]).
precondition(extinguish, [location(fire), raging(fire), equipped(team),
          deenergized(fire, area), set(boundaries), confronted(fire),
          not(dead(casualty))]).
precondition(set(reflash, watch), [location(fire), not(watched(reflashing)),
          verified(out(fire)), safe(gases), safe(oxygen)]).
precondition(verify(out), [location(fire), out(fire)]).
precondition(test(gases), [location(fire), out(fire), equipped(team)]).
precondition(test(oxygen, tester), [equipped(team)]).
precondition(test(oxygen), [location(fire), out(fire), tested(oxygen, tester)]).
precondition(desmoke, [location(fire), out(fire), smokey]).
precondition(estimate(water), [location(fire), watery]).
precondition(dewater, [location(fire), watery, estimated(water)]).
precondition(store(equipment), [location(repair, locker), equipped(team)]).
precondition(debrief, [location(repair, locker), not(equipped(team)),
          watched(reflashing)]).
precondition(secure(reflash, watch), [watched(reflashing), debriefed(team)]).
precondition(go(fire), [location(repair, locker), not(dead(casualty))]).
precondition(go(repair, locker), [location(fire), not(dead(casualty))]).
precondition(wait, []).

deletepostcondition(remove(casualty), [present(casualty), treated(casualty)]).
deletepostcondition(direct(medical, corpman), []).
deletepostcondition(give(first, aid), []).
deletepostcondition(replace(casualty), [unreplaced(casualty)]).
deletepostcondition(equip, []).
deletepostcondition(deenergize, [energized, not(smokey), tested(gases),
          tested(oxygen), debriefed(team), verified(out(fire))]).
deletepostcondition(set(boundaries), [tested(gases), tested(oxygen), debriefed(team),
          verified(out(fire)), confronted(fire)]).
deletepostcondition(approach(fire), [tested(gases), tested(oxygen), debriefed(team),
          verified(out(fire))]).
deletepostcondition(extinguish, [raging(fire), tested(gases), tested(oxygen),
          verified(out(fire)), watched(reflashing), debriefed(team), set(boundaries),
          safe(gases), safe(oxygen), unsafe(gases), unsafe(oxygen), confronted(fire)]).
deletepostcondition(set(reflash, watch), []).
deletepostcondition(verify(out), []).
deletepostcondition(test(gases), [unsafe(gases), safe(gases)]).
deletepostcondition(test(oxygen, tester), []).
deletepostcondition(test(oxygen), [unsafe(oxygen), safe(oxygen)]).
deletepostcondition(desmoke, [smokey, debriefed(team), unsafe(gases), tested(gases),
          unsafe(oxygen), tested(oxygen)]).
deletepostcondition(estimate(water), []).
deletepostcondition(dewater, [watery, estimated(water), debriefed(team), tested(gases),
          unsafe(gases), tested(oxygen), unsafe(oxygen)]).
deletepostcondition(store(equipment), [equipped(team)]).
deletepostcondition(debrief, []).
deletepostcondition(secure(reflash, watch), [watched(reflashing)]).
deletepostcondition(go(fire), [location(repair, locker)]).
deletepostcondition(go(repair, locker), [location(fire), confronted(fire), tested(gases),
        tested(oxygen)]).
deletepostcondition(wait, [tested(gases), tested(oxygen), unsafe(gases),
        unsafe(oxygen)]).

addpostcondition(remove(casualty), [unreplaced(casualty)]).
addpostcondition(direct(medical, corpman), [treated(casualty)]).
addpostcondition(give(first, aid), [treated(casualty)]).
addpostcondition(replace(casualty), []).
addpostcondition(equip, [equipped(team)]).
addpostcondition(deenergize, [deenergized(fire, area), smokey]).
addpostcondition(set(boundaries), [set(boundaries), smokey]).
addpostcondition(approach(fire), [confronted(fire), smokey]).
addpostcondition(extinguish, [out(fire), watery, smokey]).
addpostcondition(set(reflash, watch), [watched(reflashing)]).
addpostcondition(verify(out), [verified(out(fire))]).
addpostcondition(test(gases), [tested(gases), safe(gases)]).
addpostcondition(test(oxygen, tester), [tested(oxygen, tester)]).
addpostcondition(test(oxygen), [tested(oxygen), safe(oxygen)]).
addpostcondition(desmoke, []).
addpostcondition(estimate(water), [estimated(water)]).
addpostcondition(dewater, []).
addpostcondition(store(equipment), []).
addpostcondition(debrief, [debriefed(team)]).
addpostcondition(secure(reflash, watch), []).
addpostcondition(go(fire), [location(fire)]).
addpostcondition(go(repair, locker), [location(repair, locker)]).
addpostcondition(wait, []).

randsubst(equip, [[none, present(medical, corpman), 0.5]]).
randsubst(approach(fire), [[none, present(casualty), 0.15, 'A team member got burned.']]).
randsubst(extinguish, [[out(fire), raging(fire), 0.3, 'Fire is still raging.'],
        [none, present(casualty), 0.15, 'A team member got burned.']]).
randsubst(verify(out), [[out(fire), raging(fire), 0.2, 'Unfortunately the fire has flared
        up again.']]).
randsubst(test(gases), [[safe(gases), unsafe(gases), 0.2, 'The gases are unsafe.']]).
randsubst(test(oxygen), [[safe(oxygen), unsafe(oxygen), 0.2, 'The oxygen is unsafe.']]).
randsubst(desmoke, [out(fire), raging(fire), 0.1, 'The fire flared up again.'],
        [none, present(casualty), 0.1, 'A team member is injured.']]).
randsubst(estimate(water), [out(fire), raging(fire), 0.05, 'The fire flared up again.'],
        [none, present(casualty), 0.1, 'A team member is injured.']]).
randsubst(dewater, [[out(fire), raging(fire), 0.1, 'The fire flared up again.'],
        [none, present(casualty), 0.1, 'A team member is injured.']]).
randsubst(store(equipment), [[out(fire), raging(fire), 0.05, 'The fire flared up again.']]).
randsubst(debrief, [[out(fire), raging(fire), 0.05, 'The fire flared up again.']]).

deletepostcondition(deenergize, [not(equipped(team))], [tested(gases), tested(oxygen),
        verified(out(fire))]).
deletepostcondition(debrief, [location(fire)], []).

addpostcondition(deenergize, [not(equipped(team))], [present(casualty)]).
addpostcondition(approach(fire), [not(equipped(team))], [present(casualty), smokey]).
addpostcondition(test(gases), [raging(fire)], [tested(gases), unsafe(gases)]).
addpostcondition(test(oxygen), [raging(fire)], [tested(oxygen), unsafe(oxygen)]).

addpostcondition(O, [present(casualty)], [dead(casualty)|APL], 'Your casualty died!') :-
        singlemember(O, [go(repair, locker), go(fire), equip, deenergize,
        set(boundaries), approach(fire), extinguish, desmoke, estimate(water),
        dewater, test(gases), test(oxygen, tester), test(oxygen), set(reflash, watch),
        store(equipment), debrief, secure(reflash, watch)]),
        addpostcondition(O, APL).
nopref(set(boundaries), deenergize).
nopref(test(oxygen, tester), test(gases)).
nopref(desmoke, dewater).
nopref(desmoke, estimate(water)).

start_state([location(repair, locker), raging(fire), smokey]).
goal([verified(out(fire)), safe(gases), safe(oxygen), not(equipped(team)), not(smokey),
        not(watery), not(watched(reflashing)), not(present(casualty)),
        not(unreplaced(casualty)), not(dead(casualty)), debriefed(team)]).

/* Version of "go" for versions 20 and below of the tutor */
go2 :- start_state(S), goal(G), tutor(S, G).
```

REFERENCES

[BOWE 91] Bowen,Kenneth A., *Prolog and Expert Systems*, McGraw-Hill, Inc., 1991

[ROWE 88] Rowe, Neil C., *Artificial Intelligence through Prolog*, Prentice-Hall, Inc. 1988

[TURB 90] Turban, Efraim, *Decision Support and Expert Systems: Management Support Systems*, Macmillan Publishing Company, 1990

[SMIT 91] Smith, David N., *Concept of Object Oriented Programming*, McGraw-Hill, Inc., 1991

[BRAT 91] Bratko, Ivan, *PROLOG, Programming for Artificial Intelligence*, Addison-Wesley Publishing Co., 1991

[HEND 88] Hendler, James A., *Expert Systems: The User Interface*, Ablex Publishing Corporation, 1988

[WALK 87] Walker, Adrian I., *Knowledge Systems and Prolog*, Addison-Wesley Publishing Company, Inc., 1987.

[BOOT 89] Booth, Paul A., *An Introduction to Human-Computer Interaction*, Lawrence Erlbaum Associates, 1989

[CARR 87] Carrol, John M., *Interfacing Thought*, The MIT Press, 1987

[SHNE 87] Shneiderman, Ben, *Designing the User Interface*, Addison-Wesley Publishing Company, 1987

[KLAH 86] Klahr, Philip and Donald A. Waterman, *Expert Systems Techniques, Tools and Applications*, Addison-Wesley Publishing Company, 1986

[CLANC 87] Clancey, William J., *Knowledge-Based Tutoring: The GUIDON Program*, The MIT Press, 1987

[BROW 89] Brown, David C. and B. Chandrasekaran, *Design Problem Solving: Knowledge Structures and Control Strategies*, Pitman Publishing, 1989

[GUID 89] Guida, Giovanni and Carlo Tasso, *Topics in Expert System Design: Methodologies and Tools*, Elsevier Science Publishers B. V., 1989

[COVI 88] Covington, Michael A., Donald Nute and Andre Vellino, *Prolog Programming in Depth*, Scott, Foresman and Company, 1988

[WALK 90] Walker, Terry C. and Richard K. Miller, CMfgE, CEM, *Expert Systems Handbook*, The Fairmont Press Inc., 1990

[BROW1 89]Brown, Judith R. and Steve Cunningham, *Programming the User Interface*, John Willey & Sons, Inc., 1989

[BAIL 82] Bailey, Robert W., *Human Performance Engineering: A Guide for System Designers*, Prentice-Hall, Inc, 1982

[MORA 81] Moran, T. P. , The Command Language Grammar: A representation for the user inerface of interactive computer systems, *International Journal of Man-Machine Studies*, 15, 3-50, 1981

[MCMIL 85] McMillan, T. C. and D. A. Moran, Command Line Structure and Dinamic Processing of Abbreviations in Dialog Management, *Interfaces in Computing, 3*, 1985

[QUIN 90] Quintus Computer Systems, Inc., *Quintus Prolog Reference Manual*, 1990

[QUIN 88] Quintus Computer System, Inc., *Quintus ProWINDOWS User's Guide*, 1988

[KEAR 87] Kearsley, G. P., *Artificial Intelligence & Instruction*, Addison-Wesley, 198

[ROWE 90] Rowe, Neil C., "Means-Ends Tutoring, Multi-Tutoring and Meta-Tutoring", Naval Postgraduate School, Report NPS52-90-003, p. 9, February 1990

[PSOT 88] Psotka, Josesph, L. Dan Massey and Sharon A. Mutter, *Intelligent Tutoring Systems, Lesson Learned*, Lawrence Erlbraum Associates Publishers, 1988

[SLEE 82] Sleeman, D. and Brown, J. S., *Intelligent Tutoroing Systems*, Academic Press, 1982

[CAMP 88] Campbell, D. S., "An Intelligent Computer-Assisted Instruction System for Cardiopulmonary Resuscitation", M.S. thesis, Department of Computer Science, U. S. Naval Postgraduate School, Monterey CA, June 1988

[KIM 88] Kim, T. W., "A Computer-Aided Instruction Program for Teaching the TOP20-MM Facility on the DDN", M.S. thesis, U. S. Naval Postgraduate School, Monterey CA, June 1988

[SALG 89] Salgado-Zapata, P. J., "An Intelligent Computer-Assisted Instruction System for Underway Replenihsment", M.S. thesis, U. S. Naval Postgraduate School, Monterey CA, June 1989

[WEIN 88] Weingart, S. G., "Development of a Shipboard Damage Control Fire Team Leader Intelligent Computer-Aided Instructional Tutoring System", M.S. thesis, Department of Computer Science, U. S. Naval Postgraduate School, Monterey CA, June 1988

[KANG 90] Kang, Moung-Hung, "Pilot emergency Tutoring System for F-4 Aircraft Fuel System Malfunction Using Means-Ends Analysis", M.S. thesis, Department of Computer Science, U. S. Naval Postgraduate School, Monterey CA, June 1990

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexanderia, VA 22304-6145	2
2.	Dudley Knox Library Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Dr. Neil C.Rowe Code CSRp Assistant Professor, Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Dr. Timothy J. Shimeall Code CSSm Assistant Professor, Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	1
5.	Kepala Staf Umum ABRI Mabes ABRI, Cilangkap, Jakarta Timur, 13870, Indonesia	1
6.	Sekjen Dephankam Departemen Pertahanan Keamanan Jl. Merdeka Barat No. 13-14, Jakarta Indonesia	1
7.	Depers KASAU Mabes TNI-AU, Jl. Gatot Subroto No. 72, Jakarta Timur, Indonesia	1
8.	Diraeroau Mabes TNI-AU, Jl. Gatot Subroto No. 72, Jakarta Timur, Indonesia	1
9.	Office of Defense Attache Embassy of the Republic of Indonesia 2020 Massachusetts Avenue, N.W. Washington, D.C., 20036	1

10. LTC Francius Suwono 2
Jl. Sengkuni 2, Dirgantara 3,
Halim Pk, Jakarta Timur 13610
Indonesia